Application Session Filtering Cookbook

DOCUMENT REVISION – 9/25/15

Contents

Chapter 1

Introduction

This guide presents a number of examples of how to use Application Session Filtering (ASF) in step-by-step recipes. These recipes are described in Chapter 2, *Application Session Filtering Recipes*.

This chapter provides a methodology for identifying string patterns and regular expressions. ASF uses the pattern-matching and regular expression engine in Adaptive Packet Filtering (APF) to select packet flows based on matching criteria with one or more packets in the flow session. Buffering of session initialization handshakes ensures that the entire session, from start to finish, is either dropped or forwarded to the security or performance monitoring tools.

The section *Getting Started* on page 8 provides some basic information that you may want to review before using the commands in Chapter 2, *Application Session Filtering Recipes*.

Application Session Filtering (ASF) is available with GigaVUE-OS 4.4. For further details about ASF and APF, refer to the *GigaVUE-OS CLI User's Guide* for GigaVUE-OS 4.4 or later.

NOTE: GigaVUE-OS 4.4 or later is required to use the CLI commands presented in this document.

ASF and the GigaSECURE Security Delivery Platform

The GigaSECURE® security delivery platform connects into both physical and virtual networks, and can be configured to deliver traffic to all of the applications that require it. Security appliances connect into the platform at whatever interface speeds they are capable of to receive a stream of relevant traffic from across the network infrastructure. The GigaSECURE platform also extracts flow-based metadata from network traffic, which can be routed to various security tools for analysis.

Application Session Filtering (ASF) with or without buffering is one of the pillars of the GigaSECURE security deliver platform. ASF provides the ability to deliver just the relevant traffic streams to specific types of security tools by steering entire sessions to a specific security solution or discarding the entire session so large volumes of irrelevant data is not processed. For example, an email security solution need not see YouTube traffic. Sending only relevant

traffic allows the security solutions to function more effectively and waste less bandwidth and resources processing irrelevant information.

Many security solutions do not need to look at entire flows that are either trusted or that they have no ability to process. ASF provides the ability to look deep into the packet at the application layer, identify application flows based on patterns within the packets, and steer entire sessions to a specific security solution (for example, all packets belonging to a session, even if subsequent or preceding packets for that session do not match the pattern) or to discard the entire session.

This powerful capability allows precise control of the types of traffic data sent to security tools based on Layer 4 to Layer 7 and more sophisticated content matching, thereby ensuring that security solutions are focused on working off network traffic that is most relevant to them while simultaneously offloading those tools from having to process large volumes of irrelevant data.

ASF is just one pillar in the GigaSECURE security delivery platform. SSL decryption is another. While ASF makes it possible to steer entire sessions to a specific security solution or discard the entire session, SSL decryption allows tools to peek into encrypted channels of communication. ASF can be combined with the SSL decryption as part of a single solution where encrypted traffic is decrypted and then filtered. An example of combining SSL decryption and ASF is described in the section *Filtering Traffic with SSL Decryption* on page 22. This cookbook also describes many more examples of how to use ASF. This provides additional support for the GigaSECURE security delivery platform with ready-made *recipes* for filtering common types of traffic, such as Netflix, YouTube, and so on.

Methodology for Identifying String Patterns and Regular Expressions

The recipes in the next chapter use regular expressions (RegEX) and strings patterns in the second level maps to identify the traffic to drop or pass during the session. You can use the regular expression or strings or both in the recipes. However, you may want to develop your own. If you want to develop your own, the following are some steps to follow to identify regular expressions or strings for use in a recipe:

1. Decide what type of traffic either to pass or drop so it is not passed to your monitoring tool or tools. A few examples include the following:
   - All of a particular layer 7 protocol. For example, Email/SMTP, HTTPS, or HTTP.
   - Only traffic to or from a particular website, such as Netflix, Pandora, Amazon, or an internal SQL server.
   - Only a particular content type, such as music, videos, or images.
2. Get a complete “user session” sample of the traffic including the TCP handshake or handshakes of all TCP connections that would or could make up the user session. The following are a some examples:
   - There are generally multiple TCP connections and sources involved in populating one user request for a Web page. Live HTTP headers are a useful tool for helping understand how many TCP connections are involved in one Web page request.
   - Capturing filters such as the source and destination IP addresses of a PC are helpful in reducing the amount of capture traffic to analyze.
3. Using a packet analysis tool, such as Wireshark, isolate the TCP connections in a packet capture. If you use Wireshark, you can do either of the following:
   - Isolate the TCP sessions with the following display filter:

     ```
     tcp.flags.syn == 1 && tcp.flags.ack == 0
     ```
   - Use the **Find Packet** tool to perform a String filter on the packet bytes for different aspects of the traffic that you are trying to isolate, such as domain names and content types. Figure 1-1 shows an example of the Find Packet tool with a String filter on the “content-type”.

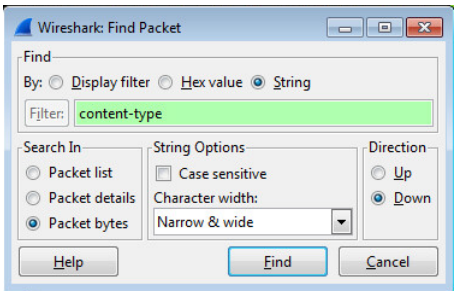

Figure 1-1: Example of a Find Packet Tool

4. Follow each TCP stream to find clear text that is relevant to the traffic that you want to isolate.

Some resources that may help are HTTP request headers and their fields, Content-Type for different protocols, and understanding the structure and data in an SSL certificate.

5. Once the clear text is isolated, decide what gives the least number of false positive matches. Use these clear text strings to create the regular expression or strings (or both) to match. There are many resources on the Internet that can help with this portion of the task.
6. Use the regular expression or strings (or both) from the previous step to determine what to use in the second level map.

   ASF is a two part process that uses a first level and second level map. The second level map uses the regular expression. As a method for lowering the number of false positives, the first level map can be based on Layer 2 through 4 parameters to narrow down the type of traffic that is sent to the second level map.

Getting Started

To use Application Session Filtering (ASF) and Adaptive Packet Filtering (APF), you need to have the appropriate licenses.

Once you have accessed the command line interface (CLI), you need to put the CLI into Configuration mode by doing the following:

1. From Standard mode, enter **en** at the > prompt to get to Enable mode. The prompt now changes to a hashtag.
2. Next, enter **`config t`** at the Enable mode prompt. The prompt now changes to `(config)#`.

Application Session Filtering Recipes

This chapter describes a number of recipes for filtering high-volume traffic, such as video streaming, preventing this high-volume traffic from overloading malware detectors and other security tools. In other recipes, only video or audio traffic might be filtered for monitoring tools.

The following recipes are presented in this chapter:

NOTE: The first time you configure an ASF buffer, you need to restart the GigaSMART engine using the following commands. The first command brings down the engine. The second command brings it back up.

```
card slot [slot ID] down
no card slot [slot ID] down
```

Filtering Netflix Traffic

These recipes described in this section provide the steps for filtering all Netflix traffic sessions, either dropping or passing the session.

Dropping Netflix Traffic

This recipe drops all Netflix traffic sessions and passes all other traffic to the existing tools.

To pass Netflix traffic, refer to *Passing Netflix Traffic* on page 11.

To set up this recipe, perform the following steps:

Step	Description	Command
1.	Configure the network port.	**(config) # port 1/1/g1..g4 params admin enable**
2.	Configure the tool port and enable it.	**(config) # port 1/1/g5 type tool** **(config) # port 1/1/g5 params admin enable**
3.	Configure a GigaSMART group and associate it with GigaSMART engine ports.	**(config) # gsgroup alias gsgrp1 port-list 1/1/e1**
4.	Define the maximum number of sessions. (The '2' indicates a maximum of 2 million sessions.)	**(config) # gsparams gsgroup gsgrp1 resource buffer-asf 2**
5.	Reload the GigaSMART card. **NOTE:** The card only needs to be reloaded the first time an ASF buffer is created.	**(config) # card slot 1/1 down** **(config) # no card slot 1/1 down**
6.	Create a flow session, specify the buffer count before the match, and enable buffering. **NOTE:** This step sets the number of packets buffered to 20. However, after performing your own investigations, you may want to set the number of packets to a lower value to optimize performance for your particular situation.	**(config) # apps asf alias netflix-sessions** **(config apps asf alias netflix-sessions) # buffer enable** **(config apps asf alias netflix-sessions) # bi-directional enable** **(config apps asf alias netflix-sessions) # buffer-count-before-match 20** **(config apps asf alias netflix-sessions) # sess-field add ipv4-5tuple outer** **(config apps asf alias netflix-sessions) # exit** **(config) #**
7.	Configure combined GigaSMART operation.	**(config) # gsop alias netflix-gsop apf set asf netflix-sessions port-list gsgrp1**
8.	Create a virtual port and associate it with the gsgroup.	**(config) # vport alias vp1 gsgroup gsgrp1**
9.	Create a first level map to filter Web traffic.	**(config) # map alias webtraffic** **(config map alias webtraffic) # rule add pass portdst 80 bidir** **(config map alias webtraffic) # rule add pass portdst 443 bidir** **(config map alias webtraffic) # to vp1** **(config map alias webtraffic) # from 1/1/g1..g4** **(config map alias webtraffic) # exit** **(config) #**

Step	Description	Command
10.	Create a second level map. The gsrule specifies the traffic to drop, using keywords. Buffered packets and all subsequent packets will be dropped.	**(config) # map alias netflix_filter** **(config map alias netflix_filter) # use gsop netflix-gsop** **(config map alias netflix_filter) # gsrule add drop pmatch protocol tcp pos 1 RegEx “netflix|nflxvideo|nflximg|Netflix|nflxext|nflxsearch” 0..1000** **(config map alias netflix_filter) # gsrule add drop pmatch protocol tcp pos 1 string “octet-stream” 0..1000** **(config map alias netflix_filter) # to 1/1/g5** **(config map alias netflix_filter) # from vp1** **(config map alias netflix_filter) # exit** **(config) #**
11.	Create a map for the collector.	**(config) # map-scollector alias collector** **(config map-scollector alias collector) # from vp1** **(config map-scollector alias collector)) # collector 1/1/g5** **(config map-scollector alias collector)) # exit** **(config) #**
12.	(Optional) Display the configuration for this example. (Output not shown.)	**(config) # show gsgroup** **(config) # show gsparams** **(config) # show apps asf all** **(config) # show gsop** **(config) # show vport** **(config) # show map**

Passing Netflix Traffic

To pass Neflix traffic, replace the commands in *Step 10.* of the recipe with the following:

(config) # map alias netflix_filter
(config map alias netflix_filter) # use gsop netflix-gsop
(config map alias netflix_filter) # gsrule add pass pmatch protocol tcp pos 1 RegEx “netflix\|nflxvideo\|nflximg\|Netflix\|nflxext\|nflxsearch” 0..1000
(config map alias netflix_filter) # gsrule add pass pmatch protocol tcp pos 1 string “octet-stream” 0..1000
(config map alias netflix_filter) # to 1/1/g5
(config map alias netflix_filter) # from vp1
(config map alias netflix_filter) # exit
(config) #

Filtering YouTube Traffic

These recipes provide the steps for filtering all YouTube traffic sessions. either dropping or passing the session.

Dropping YouTube Traffic

This recipe drops all YouTube traffic sessions and passes all other traffic to the existing tools.

To pass all YouTube traffic, refer to *Passing YouTube Traffic* on page 13.

To set up this recipe, perform the following steps:

Step	Description	Command
1.	Configure the network port.	**(config) # port 1/1/g1..g4 params admin enable**
2.	Configure the tool port and enable it.	**(config) # port 1/1/g5 type tool** **(config) # port 1/1/g5 params admin enable**
3.	Configure a GigaSMART group and associate it with GigaSMART engine ports.	**(config) # gsgroup alias gsgrp1 port-list 1/1/e1**
4.	Define the maximum number of sessions. (The '2' indicates a maximum of 2 million sessions.)	**(config) # gsparams gsgroup gsgrp1 resource buffer-asf 2**
5.	Reload the GigaSMART card. **NOTE:** The card only needs to be reloaded the first time an ASF buffer is created.	**(config) # card slot 1/1 down** **(config) # no card slot 1/1 down**
6.	Create a flow session, specify the buffer count before the match, and enable buffering. **NOTE:** This step sets the number of packets buffered to 20. However, after performing your own investigations, you may want to set the number of packets to a lower value to optimize performance for your particular situation.	**(config) # apps asf alias youtube-sessions** **(config apps asf alias youtube-sessions) # buffer enable** **(config apps asf alias youtube-sessions) # bi-directional enable** **(config apps asf alias youtube-sessions) # buffer-count-before-match 20** **(config apps asf alias youtube-sessions) # sess-field add ipv4-5tuple outer** **(config) # exit** **(config) #**
7.	Configure combined GigaSMART operation.	**(config) # gsop alias youtube-gsop apf set asf youtube-sessions port-list gsgrp1**
8.	Create a virtual port and associate it with the gsgroup.	**(config) # vport alias vp1 gsgroup gsgrp1**
9.	Create a first level map to filter Web traffic.	**(config) # map alias webtraffic** **(config map alias webtraffic) # rule add pass portdst 80 bidir** **(config map alias webtraffic) # rule add pass portdst 443 bidir** **(config map alias webtraffic) # from 1/1/g1..g2** **(config map alias webtraffic) # to vp1** **(config map alias webtraffic) # exit** **(config) #**

Step	Description	Command
10.	Create a second level map. The gsrule specifies the traffic to drop, using keywords. Buffered packets and all subsequent packets will be dropped.	**(config) # map alias youtube_filter** **(config map alias youtube_filter) # use gsop youtube-gsop** **(config map alias youtube_filter) # gsrule add drop pmatch protocol tcp pos 1 RegEx “youtube|ytimg|yt3.ggpht|tubeMogul|tmogulyoutu” 0..1000** **(config map alias youtube_filter) # gsrule add drop pmatch protocol tcp pos 1 string “Content-Type: video” 0..1000** **(config map alias youtube_filter) # gsrule add drop pmatch protocol tcp pos 1 string “octet-stream” 0..1750** **(config map alias youtube_filter) # to 1/1/x4** **(config map alias youtube_filter) # from vp1** **(config map alias youtube_filter) # exit** **(config) #**
11.	Create a map for the collector.	**(config) # map-scollector alias collector** **(config map-scollector alias collector) # from vp1** **(config map-scollector alias collector) # collector 1/1/x4** **(config map-scollector alias collector) # exit** **(config) #**
12.	(Optional) Display the configuration for this example. (Output not shown.)	**(config) # show gsgroup** **(config) # show gsparams** **(config) # show apps asf all** **(config) # show gsop** **(config) # show vport** **(config) # show map**

Passing YouTube Traffic

To pass YouTube traffic, replace the commands in *Step 10.* of the recipe with the following:

(config) # map alias youtube_filter
(config map alias youtube_filter) # use gsop youtube-gsop
(config map alias youtube_filter) # gsrule add pass pmatch protocol tcp pos 1 RegEx “youtube|ytimg|yt3.ggpht|tubeMogul|tmogulyoutu” 0..1000
(config map alias youtube_filter) # gsrule add pass pmatch protocol tcp pos 1 string “Content-Type: video” 0..1000
(config map alias youtube_filter) # gsrule add pass pmatch protocol tcp pos 1 string “octet-stream” 0..1000
(config map alias youtube_filter) # to 1/1/x4
(config map alias youtube_filter) # from vp1
(config map alias youtube_filter) # exit
(config) #

Filtering Facebook Traffic

These recipes provide the steps for filtering Facebook traffic session, either dropping or passing the session.

Dropping Facebook Traffic

This recipe identifies Facebook traffic and drops it from the session.

To pass Facebook traffic, refer to *Passing Facebook Traffic* on page 15.

To set up this recipe, perform the following steps:

Step	Description	Command
1.	Configure the network port.	**(config) # port 1/1/g1..g4 params admin enable**
2.	Configure the tool port and enable it.	**(config) # port 1/1/g5 type tool** **(config) # port 1/1/g5 params admin enable**
3.	Configure a GigaSMART group and associate it with GigaSMART engine ports.	**(config) # gsgroup alias gsgrp1 port-list 1/1/e1**
4.	Define the maximum number of sessions. (The '2' indicates a maximum of 2 million sessions.)	**(config) # gsparams gsgroup gsgrp1 resource buffer-asf 2**
5.	Reload the GigaSMART card. **NOTE:** The card only needs to be reloaded the first time an ASF buffer is created.	**(config) # card slot 1/1 down** **(config) # no card slot 1/1 down**
6.	Create a flow session, specify the buffer count before the match, and enable buffering. **NOTE:** This step sets the number of packets buffered to 20. However, after performing your own investigations, you may want to set the number of packets to a lower value to optimize performance for your particular situation.	**(config) # apps asf alias fb-sessions** **(config apps asf alias fb-sessions) # buffer enable** **(config apps asf alias fb-sessions) # bi-directional enable** **(config apps asf alias fb-sessions) # buffer-count-before-match 20** **(config apps asf alias fb-sessions) # sess-field add ipv4-5tuple outer** **(config apps asf alias fb-sessions) # exit** **(config) #**
7.	Configure combined GigaSMART operation.	**(config) # gsop alias fb-gsop apf set asf fb-sessions port-list gsgrp1**
8.	Create a virtual port and associate it with the gsgroup.	**(config) # vport alias vp1 gsgroup gsgrp1**
9.	Create a first level map to filter Web traffic.	**(config) # map alias webtraffic** **(config map alias webtraffic) # rule add pass portdst 80 bidir** **(config map alias webtraffic) # rule add pass portdst 443 bidir** **(config map alias webtraffic) # to vp1** **(config map alias webtraffic) # from 1/1/g1..g2** **(config map alias webtraffic) # exit** **(config) #**

Step	Description	Command
10.	Create a second level map. The gsrule specifies the traffic to drop, using keywords. Buffered packets and all subsequent packets will be dropped.	**(config) # map alias fb_filter** **(config map alias fb_filter) # use gsop fb-gsop** **(config map alias fb_filter) # gsrule add drop pmatch protocol tcp pos 1 RegEx "^fdcdn.*\.akamaihd\.net|fbstatic.*\.akamaihd\.net|fbexternal.*\.akamaihd\.net|.*\.facebook\.com|.*\.fbcdn\.net" 0..1750** **(config map alias fb_filter) # to 1/1/g5** **(config map alias fb_filter) # from vp1** **(config map alias fb_filter) # exit** **(config) #**
11.	Create a map for the collector.	**(config) # map-scollector alias collector** **(config map-scollector alias collector) # from vp1** **(config map-scollector alias collector) # collector 1/1/g5** **(config map-scollector alias collector) # exit** **(config) #**
12.	(Optional) Display the configuration for this example. (Output not shown.)	**(config) # show gsgroup** **(config) # show gsparams** **(config) # show apps asf all** **(config) # show gsop** **(config) # show vport** **(config) # show map**

Passing Facebook Traffic

To pass YouTube traffic, replace the commands in *Step 10.* of the recipe with the following:

(config) # map alias fb_filter
(config map alias fb_filter) # use gsop fb-gsop
(config map alias fb_filter) # gsrule add pass pmatch protocol tcp pos 1 RegEx "^fdcdn.*\.akamaihd\.net\|fbstatic.*\.akamaihd\.net\|fbexternal.*\.akamaihd\.net\|.*\.facebook\.com\|.*\.fbcdn\.net" 0..1750
(config map alias fb_filter) # to 1/1/g5
(config map alias fb_filter) # from vp1
(config map alias fb_filter) # exit
(config) #

Filtering HTTPS Traffic

These recipes provide the steps for filtering HTTPS traffic on any port or on non-standard ports, either dropping or passing the session.

Dropping HTTPS Traffic on Any TCP Port

This recipe drops HTTPS traffic that uses any Layer 4 port.

To pass HTTPS traffic, refer to *Passing HTTPS Traffic on Any TCP Port* on page 17.

To filter HTTPS traffic that uses non-standard Layer 4 ports, refer to *Filtering HTTPS Traffic on Non-Standard Ports* on page 17

To set up the recipe, perform the following steps:

Step	Description	Command
1.	Configure the network port.	**(config) # port 1/1/g1..g4 params admin enable**
2.	Configure the tool port and enable it.	**(config) # port 1/1/g5 type tool** **(config) # port 1/1/g5 params admin enable**
3.	Configure a GigaSMART group and associate it with GigaSMART engine ports.	**(config) # gsgroup alias gsgrp1 port-list 1/1/e1**
4.	Define the maximum number of sessions. (The '2' indicates a maximum of 2 million sessions.)	**(config) # gsparams gsgroup gsgrp1 resource buffer-asf 2**
5.	Reload the GigaSMART card. **NOTE:** The card only needs to be reloaded the first time an ASF buffer is created.	**(config) # card slot 1/1 down** **(config) # no card slot 1/1 down**
6.	Create a flow session, specify the buffer count before the match, and enable buffering. **NOTE:** This step sets the number of packets buffered to 20. However, after performing your own investigations, you may want to set the number of packets to a lower value to optimize performance for your particular situation.	**(config) # apps asf alias https-sessions** **(config apps asf alias https-sessions) # bi-directional enable** **(config apps asf alias https-sessions) # buffer enable** **(config apps asf alias https-sessions) # buffer-count-before-match 20** **(config apps asf alias https-sessions) # sess-field add ipv4-5tuple outer** **(config apps asf alias https-sessions) # exit** **(config) #**
7.	Configure combined GigaSMART operation.	**(config) # gsop alias https-gsop apf set asf https-sessions port-list gsgrp1**
8.	Create a virtual port and associate it with the gsgroup.	**(config) # vport alias vp1 gsgroup gsgrp1**
9.	Create a first level map to filter IPv4 traffic.	**(config) # map alias ipv4traffic** **(config map alias ipv4traffic) # rule add pass ipver 4 bidir** **(config map alias ipv4traffic) # to vp1** **(config map alias ipv4traffic) # from 1/1/g6** **(config map alias ipv4traffic) # exit** **(config) #**

Step	Description	Command
10.	Create a second level map. The gsrule specifies the traffic to drop. The RegEx expression identifies the SSL handshake type Client Hello hexadecimal patterns in an SSL TCP session. Buffered packets and all subsequent packets will be passed.	**(config) # map alias https_filter** **(map alias https_filter) # use gsop https-gsop** **(map alias https_filter) # gsrule add drop pmatch protocol tcp pos 1 RegEx "\x16\x03.{3}\x01" 0..6** **(map alias https_filter) # to 1/1/g5** **(map alias https_filter) from vp1** **(map alias https_filter) # exit** **(config) #**
11.	Create a map for the collector.	**(config) # map-scollector alias collector** **(config map-scollector alias collector) # from vp1** **(config map-scollector alias collector) # collector 1/1/g5** **(config map-scollector alias collector) # exit** **(config) #**
12.	(Optional) Display the configuration for this example. (Output not shown.)	**(config) # show gsgroup** **(config) # show gsparams** **(config) # show apps asf all** **(config) # show gsop** **(config) # show vport** **(config) # show map**

Passing HTTPS Traffic on Any TCP Port

To pass HTTPS traffic on any port, replace the commands in *Step 10.* of the previous recipe with the following:

```
(config) # map alias https_filter
(map alias https_filter) # use gsop https-gsop
(map alias https_filter) # gsrule add pass pmatch protocol tcp pos 1 RegEx  "\x16\x03.{3}\x01" 0..6
(map alias https_filter) # to 1/1/g5
(map alias https_filter) from vp1
(map alias https_filter) # exit
(config) #
```

Filtering HTTPS Traffic on Non-Standard Ports

To filter HTTPS traffic on non-standard ports, replace the commands in *Step 9.* of the previous recipe with the following:

```
(config) # map alias ipv4traffic
(config map alias ipv4traffic) # rule add drop portdst 443 bidir
(config map alias ipv4traffic) # to vp1
(config map alias ipv4traffic) # from 1/1/g6
(config map alias ipv4traffic) # exit
(config) #
```

Filtering Windows Traffic

This section provides a recipe for filtering traffic that is updating the Windows OS and a recipe for filtering traffic that is upgrading the Windows OS to Windows 10.0.

Windows Update Traffic

This recipe filters Windows update traffic that is updating the Windows OS. To filter Windows traffic that is *upgrading* the Windows OS to Windows 10.0, refer to *Filtering Windows 10.0 Upgrade Traffic* on page 19.

To set up this recipe, perform the following steps:

Step	Description	Command
1.	Configure the network port.	**(config) # port 1/1/g1..g4 params admin enable**
2.	Configure the tool port and enable it.	**(config) # port 1/1/g5 type tool** **(config) # port 1/1/g5 params admin enable**
3.	Configure a GigaSMART group and associate it with GigaSMART engine ports.	**(config) # gsgroup alias gsgrp1 port-list 1/3/e1**
4.	Define the maximum number of sessions. (The '2' indicates a maximum of 2 million sessions.)	**(config) # gsparams gsgroup gsgrp1 resource buffer-asf 2**
5.	Reload the GigaSMART card. **NOTE:** The card only needs to be reloaded the first time an ASF buffer is created.	**(config) # card slot 1/1 down** **(config) # no card slot 1/1 down**
6.	Create a flow session, specify the buffer count before the match, and enable buffering. **NOTE:** This step sets the number of packets buffered to 20. However, after performing your own investigations, you may want to set the number of packets to a lower value to optimize performance for your particular situation.	**(config) # apps asf alias winupdate-sessions** **(config apps asf alias winupdate-sessions) # sess-field add ipv4-5tuple outer** **(config apps asf alias winupdate-sessions) # buffer-count-before-match 20** **(config apps asf alias winupdate-sessions) # buffer enable** **(config apps asf alias winupdate-sessions) # exit** **(config) #**
7.	Configure combined GigaSMART operation.	**(config) # gsop alias winupdates-gsop apf set asf winupdate-sessions port-list gsgrp1**
8.	Create a virtual port and associate it with the gsgroup.	**(config) # vport alias vp1 gsgroup gsgrp1**
9.	Create a first level map to filter IPv4 traffic.	**(config) # map alias ipv4traffic** **(config map alias ipv4traffic) # rule add pass ipver 4 bidir** **(config map alias ipv4traffic) # to vp1** **(config map alias ipv4traffic) # from 1/1/g1** **(config map alias ipv4traffic) # exit** **(config) #**

Step	Description	Command
10.	Create a second level map. The gsrule specifies the traffic to pass.	**(config) # map alias winupdates_filter** **(config map alias winupdates_filter) # use gsop winupdates-gsop** **(config map alias winupdates_filter) # gsrule add pass pmatch protocol tcp pos 1 RegEx "msdownload/update/software " 0..1750** **(config map alias winupdates_filter) # gsrule add pass pmatch protocol tcp pos 1 RegEx "download.windowsupdate.com" 0..1750** **(config map alias winupdates_filter) # to 1/1/g5** **(config map alias winupdates_filter) # from vp1** **(config) #**
11.	(Optional) Display the configuration for this example. (Output not shown.)	**(config) # show gsgroup** **(config) # show gsparams** **(config) # show apps asf all** **(config) # show gsop** **(config) # show vport** **(config) # show map**

Filtering Windows 10.0 Upgrade Traffic

This recipe filters traffic that is upgrading the Windows OS to Windows 10.0. To filter Windows traffic that is *updating* the Windows OS, refer to *Filtering Windows Traffic* on page 18.

To set up this recipe, perform the following steps:

Step	Description	Command
1.	Configure the network port.	**(config) # port 1/1/g6 params admin enable**
2.	Configure the tool port and enable it.	**(config) # port 1/1/g5 type tool** **(config) # port 1/1/g5 params admin enable**
3.	Configure a GigaSMART group and associate it with GigaSMART engine ports.	**(config) # gsgroup alias gsgrp1 port-list 1/1/e1**
4.	Define the maximum number of sessions. (The '2' indicates a maximum of 2 million sessions.)	**(config) # gsparams gsgroup gsgrp1 resource buffer-asf 2**
5.	Reload the GigaSMART card. **NOTE:** The card only needs to be reloaded the first time an ASF buffer is created.	**(config) # card slot 1/1 down** **(config) # no card slot 1/1 down**
6.	Create a flow session, specify the buffer count before the match, and enable buffering. **NOTE:** This step sets the number of packets buffered to 20. However, after performing your own investigations, you may want to set the number of packets to a lower value to optimize performance for your particular situation.	**(config) # apps asf alias winupgrade-sessions** **(config apps asf alias winupgrade-sessions) # bi-drectional enable** **(config apps asf alias winupdate-sessions) # buffer-count-before-match 20** **(config apps asf alias winupgrade-sessions) # buffer enable** **(config apps asf alias winupgrade-sessions) # exit** **(config) #**
7.	Configure combined GigaSMART operation.	**(config) # gsop alias winupgrade-gsop apf set asf winupgrade-sessions port-list gsgrp1**
8.	Create a virtual port and associate it with the gsgroup.	**(config) # vport alias vp1 gsgroup gsgrp1**

Step	Description	Command
9.	Create a first level map to filter IPv4 traffic.	**(config) # map alias ipv4traffic** **(config map alias ipv4traffic) # rule add pass ipver 4 bidir** **(config map alias map11) # to vp1** **(config map alias ipv4traffic) # from 1/1/g6** **(config map alias ipv4traffic) # exit** **(config) #**
10.	Create a second level map. The gsrule specifies the traffic to pass.	**(config) # map alias winupgrade_filter** **(config map alias winupgrade_filter) # use gsop winupgrade-gsop** **(config map alias winupgrade_filter) # gsrule add pass pmatch protocol tcp pos 1 RegEx "update.*windows10.0 " 0..1750** **(config map alias winupgrade_filter) # gsrule add pass pmatch protocol tcp pos 1 string ".esd" 0..1750** **(config map alias winupgrade_filter) # to 1/1/g5** **(config map alias winupgrade_filter) # from vp1** **(config) #**
11.	(Optional) Display the configuration for this example. (Output not shown.)	**(config) # show gsgroup** **(config) # show gsparams** **(config) # show apps asf all** **(config) # show gsop** **(config) # show vport** **(config) # show map**

Filtering Emails with Attachments

This recipe passes all email traffic containing attachments.

To set up this recipe, perform the following steps:

Step	Description	Command
1.	Configure the network port.	**(config) # port 1/1/g1..g4 params admin enable**
2.	Configure the tool port and enable it.	**(config) # port 1/1/g5 type tool** **(config) # port 1/1/g5 params admin enable**
3.	Configure a GigaSMART group and associate it with GigaSMART engine ports.	**(config) # gsgroup alias gsgrp1 port-list 1/1/e1**
4.	Define the maximum number of sessions. (The '2' indicates a maximum of 2 million sessions.)	**(config) # gsparams gsgroup gsgrp1 resource buffer-asf 2**
5.	Reload the GigaSMART card. **NOTE:** The card only needs to be reloaded the first time an ASF buffer is created.	**(config) # card slot 1/1 down** **(config) # no card slot 1/1 down**
6.	Create a flow session, specify the buffer count before the match, and enable buffering. **NOTE:** This step sets the number of packets buffered to 20. However, after performing your own investigations, you may want to set the number of packets to a lower value to optimize performance for your particular situation.	**(config) # apps asf alias email-sessions** **(config apps asf alias email-sessions) # buffer enable** **(config apps asf alias email-sessions) # bi-directional enable** **(config apps asf alias email-sessions) # buffer-count-before-match 20** **(config apps asf alias email-sessions) # sess-field add ipv4-5tuple outer** **(config apps asf alias email-sessions) # exit** **(config) #**
7.	Configure combined GigaSMART operation.	**(config) # gsop alias email-gsop apf set asf email-sessions port-list gsgrp1**
8.	Create a virtual port and associate it with the gsgroup.	**(config) # vport alias vp1 gsgroup gsgrp1**
9.	Create a first level map to filter SMTP traffic.	**(config) # map alias smtptraffic** **(config map alias smtptraffic) # rule add pass portsrc 25 bidir** **(config map alias smtptraffic) # from 1/1/g6** **(config map alias smtptraffic) # to vp1** **(config map alias smtptraffic) # exit** **(config) #**
10.	Create a second level map to decrypt emails with attachments. The gsrule specifies the traffic to pass, using keywords.	**(config) # map alias email_filter** **(config map alias email_filter) # use gsop email-gsop** **(config map alias email_filter) # gsrule add pass pmatch protocol tcp pos 1 RegEx "Content-Disposition" 0..1750** **(config map alias email_filter) # to 1/1/g5** **(config map alias email_filter) # from vp1** **(config map alias email_filter) # exit** **(config) #**
11.	(Optional) Display the configuration for this example. (Output not shown.)	**(config) # show gsgroup** **(config) # show gsparams** **(config) # show apps asf all** **(config) # show gsop** **(config) # show vport** **(config) # show map**

Filtering Traffic with SSL Decryption

In some cases it is useful to drop high-bandwidth encrypted traffic and alleviate the tools from the demands of decrypting and filtering. In this scenario, a corporation is looking to protect its HTTPS server, which is defined as follows:

- The server is inside the organization.
- There is access to the server's SSL private key.
- Browser client requests can be users inside or outside the organization.

Some example use cases are: Application Performance Monitoring, Data Loss Prevention (Detection), Web Threat Detection, Security Analytics, and Forensic Traffic Capture.

The following recipe shows how to apply APF and ASF to encrypted traffic. This is accomplished with an SSL map that decrypts traffic using a private key then sends the traffic to a hybrid port. (In the recipe, traffic from a Confluence server is used as an example.) The hybrid port is used to internally loop the traffic back to a virtual port where APF and ASF filtering is performed. This is done because SSL decryption cannot be combined with APF or ASF, so separate maps are needed for each.

To set up this recipe, perform the following steps:

Step	Description	Command
1.	Install the private key and create an SSL service.	**(config) # apps ssl keychain password** **(config) # apps ssl key alias confluence_key download type private-key url sftp:<path_to_private_key>** **(config) # apps ssl service alias confluence server-ip 10.80.1.75**
2.	Configure a GigaSMART group and associate it with GigaSMART engine ports.	**(config) # gsgroup alias gsgrp1 port-list 1/5/e1**
3.	Create a GigaSMART group for SSL decryption.	**(config) # gsop alias ssl ssl-decrypt in-port any out-port auto dedup set port-list gs1** **(config) # gsparams gsgroup gsgrp1 ssl-decrypt key-map add service confluence key confluence_key** **(config) # gsparams gsgroup gsgrp1 ssl-decrypt non-ssl-traffic pass**
4.	Create a first level map to filter all traffic between a single host and the Confluence server. Port 1/3/x1 is a network port and the source of the Confluence traffic. The decrypted traffic from port 1/3/x1 is sent to 1/3/x5, which is a hybrid port.	**(config) # map alias ssl** **(config map alias ssl) # use gsop ssl** **(config map alias ssl) # rule add pass ipsrc 10.80.1.75 /32 ipdst 10.50.22.21 /32 bidir** **(config map alias ssl) # from 1/3/x1** **(config map alias ssl) # to 1/3/x5** **(config map alias ssl) # exit** **(config) #**
5.	Create a session field group.	**(config) # apps sapf alias video_sessions** **(config apps asf alias video_sessions) # sess-field add ipv4-5tuple outer** **(config apps asf alias video_sessions) # exit** **(config) #**
6.	Configure combined GigaSMART operation.	**(config) # gsop alias nb_video_asf apf set sapf video_sessions port-list gsgrp1**
7.	Define the maximum number of sessions. (The '2' indicates a maximum of 2 million sessions.)	**(config) # gsparams gsgroup gsgrp1 resource buffer-asf 2**

Step	Description	Command
8.	Reload the GigaSMART card. **NOTE:** The card only needs to be reloaded the first time an ASF buffer is created.	**(config) # card slot 1/1 down** **(config) # no card slot 1/1 down**
9.	Create a virtual port and associate it with the gsgroup.	**(config) # vport alias vp1 gsgroup gsgrp1**
10.	Create an ingress first level map.	**(config) # map alias video1** **(config map alias video1) # rule add pass portdst 80 bidir** **(config map alias video1) # from 1/3/x5** **(config map alias video1) # to vp1** **(config map alias video1) # exit** **(config) #**
11.	Create an egress second level map.	**(config) # map alias video2** **(config map alias video2) # use gsop nb_video_asf** **(config map alias video2) # gsrule add pass pmatch protocol tcp pos 1 string "video/mp4" 0..1500** **(config map alias video2) # to 1/3/x6** **(config map alias video2) # from vp1** **(config map alias video2) # exit** **(config) #**
12.	(Optional) Display the configuration for this example. (Output not shown.)	**(config) # show gsgroup** **(config) # show gsparams** **(config) # show apps asf all** **(config) # show gsop** **(config) # show vport** **(config) # show map**

Filtering VOIP Traffic (SIP-Based)

To filter and drop Voice-Over-IP (VOIP) traffic, this recipe first filters UDP packets that use a specific port range used by the VOIP system, then uses a regular expression pattern to match a range of valid bit values with the first 2 bytes of the RTP header to filter for RTP/RTCP packets, which contains the audio data.

To filter for SIP packets (call setup) and RTP/RTCP packets (audio data), perform the following steps:

Step	Description	Command
1.	Configure the network port.	**(config) # port 1/1/x1 params admin enable**
2.	Configure the tool port and enable it.	**(config) # port 1/1/x3 type tool** **(config) # port 1/1/x3 params admin enable**
3.	Create a session field group.	**(config) # apps asf alias voip_sessions** **(config apps alias voip_sessions) # buffer enable** **(config apps alias voip_sessions) # buffer-count-before-match 10** **(config apps alias voip_sessions) # protocol udp** **(config apps alias voip_sessions) # sess-field add ipv4-5tuple outer** **(config apps alias voip_sessions) # exit** **(config) #**
4.	Configure a GigaSMART group and associate it with GigaSMART engine ports.	**(config) # gsgroup alias gsgrp1 port-list 1/1/e1**
5.	Define the maximum number of sessions. (The '2' indicates a maximum of 2 million sessions.)	**(config) # gsparams gsgroup gsgrp1 resource buffer-asf 2**
6.	Reload the GigaSMART card. **NOTE:** The card only needs to be reloaded the first time an ASF buffer is created.	**(config) # card slot 1/1 down** **(config) # no card slot 1/1 down**
7.	Configure combined GigaSMART operation.	**(config) # gsop alias voip-gsop apf set asf voip_sessions port-list gsgrp1**
8.	Create a virtual port and associate it with the gsgroup.	**(config) # vport alias vp1 gsgroup gsgrp1**
9.	Create a first level map for ingress packets.	**(config) map alias voip1** **(config map alias voip1) # rule add pass portsrc 5060..5061 protocol udp bidir** **(config map alias voip1) # rule add pass portsrc 16384..32767 protocol udp bidir** **(config map alias voip1) # from 1/1/x1** **(config map alias voip1) # to vp1** **(config map alias voip1) # exit** **(config) #**
10.	Create a second level map for egress traffic.	**(config) # map alias voip2** **(config map alias voip2) # use gsop voip-gsop** **(config map alias voip2) # gsrule add pass pmatch protocol udp pos 1 string "SIP/2.0" 0..200** **(config map alias voip2) # gsrule add pass pmatch protocol udp pos 1 RegEx [\\x80-\\xBF][\\x00|\\x03-\\x12|\\x19|\\x1A|\\x1C|\\x1F-\\x22|\\x60-\\x7F|\\x80|\\x83-\\x92|\\x99|\\x9A|\\x9C|\\x9F-\\xA2|\\xC0-\\xFF] 0..2** **(config map alias voip2) # to 1/1/x3** **(config map alias voip2) # from vp1** **(config map alias voip2) # exit** **(config) #**

Step	Description	Command
11.	(Optional) Display the configuration for this example. (Output not shown.)	**(config) # show gsgroup** **(config) # show gsparams** **(config) # show apps asf all** **(config) # show gsop** **(config) # show vport** **(config) # show map**

Chapter 3

ASF with Gigamon-FM APIs

This chapter shows how to use the GigaVUE-FM APIs to implement the recipes described in Chapter 2, *Application Session Filtering Recipes*. The APIs require GigaVUE-FM 3.1 or above and GigaVUE-OS 4.4.01 or above.

NOTE: This chapter assumes that you are familiar with a programming or scripting language, such as Python, and have read the *GigaVUE-FM API Getting Started Guide*. For detailed information about the APIs, refer to the *GigaVUE-FM API Reference*. The reference is available through the **Help Topics** in the GigaVUE-FM UI.

The following recipes are presented in this chapter:

Filtering Netflix, YouTube, or Facebook Traffic

The recipes for Netflix, YouTube, and Facebook traffic use three maps: a first level map, a second level map, and a collector. The rules for filtering traffic are specified in the second level map. This means that you can implement the recipe described in the previous chapter in such a way that a program or script only needs to change the second level map where the rules for filtering traffic for Netflix, YouTube or Facebook traffic are defined.

The following are the steps a script or program uses to make requests with the GigaVUE-FM APIs to implement the recipe for filtering Netflix, YouTube, or Facebook traffic:

1. Configure the network port.

Request	Payload
PATCH <fmip>/api/v1.1/inventory/ports/3_2_x5?clusterId=<clusterId>	{ "portId" : "3/2/x5", "alias" : "", "portType" : "network", "adminStatus" : "up", }

2. Configure the tool port and enable it.

Request	Payload
PATCH <fmip>/api/v1.1/inventory/ports/3_2_x6?clusterId=<clusterId>	{ "portId" : "3/2/x6", "alias" : "", "portType" : "tool", "adminStatus" : "up", }

3. Configure a GigaSMART group and associate it with GigaSMART engine ports.

Request	Payload
POST <fmip>/api/v1.1/gsgroups?clusterId=<clusterId>	{ "alias" : "gsgrp1" "ports": ["3/2/e1"] }

4. Define the maximum number of sessions. (The ‘2’ indicates a maximum of 2 million sessions.)

Request	Payload
PUT <fmip>/api/v1.1/gsgroups/gsgrp-3_2_e1/params/saApf?clusterId=<clusterId>	{ "bufferSize": 2 }

5. If this is the first time that an ASF buffer is being configured, restart the GigaSMART engine. A script or program can restart the engine by doing either of the following:

   - Sending two PATCH requests. The fist request change the card's status to `down`. The second request changes the status to `up`, which restarts the engine.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/chassis/cards/3_2?nodeId=<nodeId>`	`{ "slotId”: "3_2", "adminStatus": "down" }`
`PATCH <fmip>/api/v1.1/inventory/chassis/cards/3_2?nodeId=<nodeId>`	`{ "slotId”: "3_2" "adminStatus”: "up" }`

   - Reboot the system.

Request
`GET <fmip>/api/v1.1/clusterConfig/reboot?clusterId=<clusterId>`

   Restarting the card is recommended. When using the reboot request, the script or program needs to pause to give the system time to restart all components.

6. Create a flow session, specify the buffer count before the match, and enable buffering.

   NOTE: This step sets the number of packets buffered to 20. However, after performing your own investigations, you may want to set the number of packets to a lower value to optimize performance for your particular situation.

Request	Payload
`POST <fmip>/api/v1.1/saApfProfiles?clusterId=<clusterId>`	`{ "alias": traffic-sessions, "sessionFields": [{ "pos": 1, "type": "ipv4FiveTuple" }], "timeout": 10, "bidi": False, "buffering": { "enabled": true, "protocol": "tcp", "bufferCountBeforeMatch": 20 } }`

7. Configure combined GigaSMART operation.

   This step creates a `gsop` object that references the `saApfProfiles` object created in the previous step.

Request	Payload
`POST <fmip>/api/v1.1/gsops?clusterId=<clusterId>`	`{ "alias": "traffic-gsop", "gsGroup": "gsgrp1", "gsApps": { "apf": { "enabled": "enabled"}, "saApf": { "saApfProfile": "traffic-sessions" } } }`

8. Create a virtual port and associate it with the gsgroup.

Request	Payload
`POST <fmip>/api/v1.1/vports?clusterId=<clusterId>`	`{ "alias": "vp1", "gsGroup": "gsgrp1" }`

9. Create a first level map to filter Web traffic.

Request	Payload
`POST <fmip>/api/v1.1/maps?clusterId=<clusterId>`	`{ "alias" : "webtraffic", "type" : "firstLevel", "subType" : "byRule", "srcPorts" : ["3/2/x5"], "dstPorts" : ["vp1"], "order" : 2, "rules" : { "passRules" : [{ "ruleId" : 1, "bidi" : true, "matches" : [{ "type" : "portDst", "value" : 80 }] }, { "ruleId" : 2, "bidi" : true, "type" : "portDst", "value" : 443 }] } }`

10. Create a second level map. The gsrule specifies the traffic to drop, using keywords. Buffered packets and all subsequent packets will be dropped.

Request	Payload
`POST <fmip>/api/v1.1/maps?clusterId=<clusterId>`	For Netflix traffic, refer to *Payload for Dropping Netflix Traffic* on page 31 For YouTube traffic, refer to *Payload for Dropping YouTube Traffic* on page 32 For YouTube traffic, refer to *Payload for Dropping Facebook Traffic* on page 33 To pass the Netflix, YouTube, or Netflix traffic, refer to *Payloads for Passing Netflix, YouTube, or Facebook Traffic* on page 33

11. Create a map for the collector.

Request	Payload

`POST <fmip>/api/v1.1/maps?clusterId=<clusterId>`	`{`
	`"alias" : "collector",`
	`"type" : "secondLevel",`
	`"subType" : "collector",`
	`"srcPorts" : [ "vp1" ],`
	`"dstPorts" : [ "3/2/x6" ],`
	`}`

12. (Optional) Display the configuration for this example. (Output not shown.)

Request

`GET <fmip> api/v1.1/gsgroups/gsgrp1?clusterId=<clusterId>`
`GET <fmip>/api/v1.1/apps/saApfProfiles/netflix-sessions?clusterId=<clusterId>`
`GET <fmip>/api/v1.1/gsop/traffic-gsop?clusterId=<clusterId>`
`GET <fmip>/api/v1.1/maps/webtraffic?clusterId=<clusterId>`
`GET <fmip>/api/v1.1/maps/netflix_filter?clusterId=<clusterId>`
`GET <fmip>/api/v1.1/maps/collector?clusterId=<clusterId>`

Payload for Dropping Netflix Traffic

To drop Netflix traffic, use the following payload in *Step 10.*

```
{
   "alias" : "netflix_filter",
   "type" : "secondLevel",
   "subType" : "byRule",
   "srcPorts" : [ "vp1" ],
   "dstPorts" : [ "3/2/x6" ],
   "gsop" : "traffic-gsop",
   "gsRules" : {
      "dropRules" : [ {
      "ruleId" : 1,
      "matches" : [ {
         "type" : "pmatch",
         "value" : "netflix|nflxvideo|nflximg|Netflix|nflxext|nflxsearch",
         "matchType" : "regex",
         "matchOffset" : {
            "offsetStart" : 0,
            "protocol" : {
               "protocol" : "tcp",
               "pos" : 1
            },
```

```
            "offsetEnd" : 1000
         }
      } ]
   }, {
      "ruleId" : 2,
      "matches" : [ {
         "type" : "pmatch",
         "value" : "octet-stream",
         "matchType" : "string",
         "matchOffset" : {
            "offsetStart" : 0,
            "protocol" : {
               "protocol" : "tcp",
               "pos" : 1
            },
            "offsetEnd" : 1000
         }
      } ]
   }
}
```

Payload for Dropping YouTube Traffic

To drop YouTube traffic, use the following payload in *Step 10.*

```
{
   "alias" : "youtube_filter",
   "type" : "secondLevel",
   "subType" : "byRule",
   "srcPorts" : [ "vp1" ],
   "dstPorts" : [ "3/2/x6" ],
   "gsop" : "traffic-gsop",
   "gsRules" : {
      "dropRules" : [ {
         "ruleId" : 1,
            "matches" : [ {
               "type" : "pmatch",
               "value" : "youtube|ytimg|yt3.ggpht|tubeMogul|tmogulyoutu",
               "matchType" : "regex",
               "matchOffset" : {
                  "offsetStart" : 0,
                  "protocol" : {
                     "protocol" : "tcp",
                     "pos" : 1
                  },
                  "offsetEnd" : 1000
               }
            } ]
         }, {
            "ruleId" : 2,
               "matches" : [ {
                  "type" : "pmatch",
                  "value" : "Content-Type: video",
                  "matchType" : "string",
                  "matchOffset" : {
                     "offsetStart" : 0,
                     "protocol" : {
                        "protocol" : "tcp",
                        "pos" : 1
                     },
                     "offsetEnd" : 1000
                  }
            } ]
         }, {
            "ruleId" : 3,
            "matches" : [ {
               "type" : "pmatch",
               "value" : "octet-stream",
               "matchType" : "string",
               "matchOffset" : {
```

```
                  "offsetStart" : 0,
                  "protocol" : {
                     "protocol" : "tcp",
                     "pos" : 1
                  },
                  "offsetEnd" : 1000
               }
            } ]
         } ],
      }
   }
}
```

Payload for Dropping Facebook Traffic

To drop Facebook traffic, use the following payload in *Step 10.*

```
{
   "alias" : "fb_filter",
   "type" : "secondLevel",
   "subType" : "byRule",
   "srcPorts" : [ "vp1" ],
   "dstPorts" : [ "3/2/x6" ],
   "gsop" : "traffic-gsop",
   "gsRules" : {
      "dropRules" : [ {
         "ruleId" : 1,
         "matches" : [ {
            "type" : "pmatch",
            "value" :
"^fdcdn.*\\.akamaihd\\.net|fbstatic.*\\.akamaihd\\.net|fbexternal.*\\.akamaihd\\.net|.*
\\.facebook\\.com|.*\\.fbcdn\\.net",
            "matchType" : "regex",
            "matchOffset" : {
               "offsetStart" : 0,
               "protocol" : {
                  "protocol" : "tcp",
                  "pos" : 1
               },
               "offsetEnd" : 1000
            }
         } ]
      } ],
   }
}
```

Payloads for Passing Netflix, YouTube, or Facebook Traffic

To pass Netflix, YouTube, or Facebook traffic, change `"dropRules"` to `"passRules"` in the payloads described in the previous sections.

Filtering HTTPS Traffic

This recipe provides the steps for filtering HTTPS traffic on any port or on non-standard ports.

Dropping HTTPS Traffic on Any TCP Port

This recipe drops HTTPS traffic that uses any Layer 4 port.

To pass HTTPS traffic, refer to *Passing HTTPS Traffic on Any TCP Port* on page 38.

To filter HTTPS traffic that uses non-standard Layer 4 ports, refer to *Filtering HTTPS Traffic on Non-Standard Ports* on page 38.

To set up the recipe, perform a script or program performs the following steps:

1. Configure the network port.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/ports/3_2_x5?clusterId=<clusterId>`	`{ "portId" : "1/1/g6", "alias" : "", "portType" : "network", "adminStatus" : "up", }`

2. Configure the tool port and enable it.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/ports/3_2_x6?clusterId=<clusterId>`	`{ "portId" : "1/1/g5", "alias" : "", "portType" : "tool", "adminStatus" : "up", }`

3. Configure a GigaSMART group and associate it with GigaSMART engine ports.

Request	Payload
`POST <fmip>/api/v1.1/gsgroups?clusterId=<clusterId>`	`{ "alias" : "gsgrp1" "ports": [ "1/1/e1" ] }`

4. Define the maximum number of sessions. (The ‘2’ indicates a maximum of 2 million sessions.)

Request	Payload
`PUT <fmip>/api/v1.1/gsgroups/gsgrp1/params/saApf?clusterId=<clusterId>`	`{ "bufferSize": 2 }`

5. If this is the first time that an ASF buffer is being configured, restart the GigaSMART engine. A script or program can restart the engine by doing either of the following:
   - Send two PATCH requests. The fist request change the card’s status to `down`. The second request changes the status to `up`, which restarts the engine.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/chassis/cards/3_2?nodeId=<nodeId>`	`{ "slotId”: "3_2", "adminStatus": "down" }`
`PATCH <fmip>/api/v1.1/inventory/chassis/cards/3_2?nodeId=<nodeId>`	`{ "slotId”: "3_2" "adminStatus”: "up" }`

   - Reboot the system.

Request
`GET <fmip>/api/v1.1/clusterConfig/reboot?clusterId=<clusterId>`

   Restarting the card is recommended. When using the reboot request, the script or program needs to pause to give all components time to reinitialize.

6. Create a flow session, specify the buffer count before the match, and enable buffering.

Request	Payload
`POST <fmip>/api/v1.1/saApfProfiles?clusterId=<clusterId>`	`{ "alias": http-sessions, "sessionFields": [{ "pos": 1, "type": "ipv4FiveTuple" }], "bidi": False, "buffering": { "enabled": true, "protocol": "tcp", "bufferCountBeforeMatch": 20 } }`

NOTE: This step sets the number of packets buffered to 20. However, after performing your own investigations, you may want to set the number of packets to a lower value to optimize performance for your particular situation.

7. Configure combined GigaSMART operation.

   This step creates a `gsop` object that references the `saApfProfiles` object created in the *Step 6.*

Request	Payload
`POST <fmip>/api/v1.1/gsops?clusterId=<clusterId>`	`{ "alias": "http-gsop", "gsGroup": "gsgrp1", "gsApps": { "apf": { "enabled": "enabled"}, "saApf": { "saApfProfile": "https-sessions" } } }`

8. Create a virtual port and associate it with the gsgroup.

Request	Payload
`POST <fmip>/api/v1.1/vports?clusterId=<clusterId>`	`{ "alias": "vp1", "gsGroup": "gsgrp1" }`

9. Create a first level map to handle IPv4 traffic.

Request	Payload
`POST <fmip>/api/v1.1/maps?clusterId=<clusterId>`	`{ "alias" : "ipv4traffic", "type" : "firstLevel", "subType" : "byRule", "srcPorts" : [ "1/1/g6" ], "dstPorts" : [ "vp1" ], "rules" : { "passRules" : [ { "ruleId" : 1, "bidi" : true, "matches" : [ { "type" : "ip4Proto", "value" : 4 } ] } ] } }`

10. Create a second level map. The gsrule specifies the traffic to pass. The RegEx expression identifies the SSL handshake type Client Hello hexadecimal patterns in an SSL TCP session. Buffered packets and all subsequent packets will be passed.

Request	Payload
POST <fmip>/api/v1.1/ maps?clusterId=<clusterId>	{ "alias" : "https_filter", "type" : "secondLevel", "subType" : "byRule", "srcPorts" : ["vp1"], "dstPorts" : ["1/1/g5"], "gsop" : "https-gsop", "gsRules" : { "passRules" : [{ "ruleId" : 1, "matches" : [{ "type" : "pmatch", "value" : "\x16\x03.{3}\x01", "matchType" : "regex", "matchOffset" : { "offsetStart" : 0, "protocol" : { "protocol" : "tcp", "pos" : 1 }, "offsetEnd" : 6 } }] } }

11. Create a map for the collector.

Request	Payload
POST <fmip>/api/v1.1/ maps?clusterId=<clusterId>	{ "alias" : "collector", "type" : "secondLevel", "subType" : "collector" "srcPorts" : ["vp1"], "dstPorts" : ["1/1/g5"] }

12. (Optional) Display the configuration for this example. (Output not shown.)

Request
GET <fmip> api/v1.1/gsgroups/gsgrp1?clusterId=<clusterId> GET <fmip>/api/v1.1/apps/saApfProfiles/https-sessions?clusterId=<clusterId> GET <fmip>/api/v1.1/gsop/https-gsop?clusterId=<clusterId> GET <fmip>/api/v1.1/maps/ipv4traffic?clusterId=<clusterId> GET <fmip>/api/v1.1/maps/https_filter?clusterId=<clusterId> GET <fmip>/api/v1.1/maps/collector?clusterId=<clusterId>

Passing HTTPS Traffic on Any TCP Port

To pass HTTPS traffic on any port, change `"dropRule"` to `"passRule"` in the payload shown in *Step 10.*

Filtering HTTPS Traffic on Non-Standard Ports

To filter HTTPS traffic on non-standard ports, use the following payload for the first level map in *Step 9.*

```
{
   "alias" : "ipv4traffic",
   "type" : "firstLevel",
   "subType" : "byRule",
   "srcPorts" : [ "1/1/g6" ],
   "dstPorts" : [ "vp1" ],
   "rules" : {
      "dropRules" : [ {
         "ruleId" : 1,
         "bidi" : true,
         "matches" : [ {
            "type" : "portDst",
            "value" : 443
         } ]
      } ]
   }
}
```

Filtering Windows Traffic

This section provides a recipe for filtering traffic that is updating the Windows OS and a recipe for filtering traffic that is upgrading the Windows OS to Windows 10.0.

Filtering Windows Update Traffic

This recipe filters Windows update traffic that is updating the Windows OS. For filtering traffic that is upgrading the Windows OS to Windows 10, refer to *Filtering Windows 10.0 Upgrade Traffic* on page 43.

To set up this recipe, a program or script performs the following steps:

1. Configure the network port.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/ports/1_1_g5?clusterId=<clusterId>`	`{ "portId" : "3/2/x5", "alias" : "", "portType" : "network", "adminStatus" : "up", }`

2. Configure the tool port and enable it.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/ports/3_2_x6?clusterId=<clusterId>`	`{ "portId" : "3/2/x6", "alias" : "", "portType" : "tool", "adminStatus" : "up", }`

3. Configure a GigaSMART group and associate it with GigaSMART engine ports.

Request	Payload
`POST <fmip>/api/v1.1/gsgroups`	`{ "alias" : "gsgrp1" "ports": [ "3/2/e1" ] }`

4. Define the maximum number of sessions. (The ‘2’ indicates a maximum of 2 million sessions.)

Request	Payload
`PUT <fmip>/api/v1.1/gsgroups/gsgrp1/params/saApf`	`{ "bufferSize": 2 }`

5. If this is the first time that an ASF buffer is being configured, restart the GigaSMART engine. A script or program can restart the engine by doing either of the following:

   a. Send two PATCH requests. The fist request change the card's status to `down`. The second request changes the status to `up`, which restarts the engine.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/chassis/cards/3_2?nodeId=<nodeId>`	`{ "slotId”: "3_2", "adminStatus": "down" }`
`PATCH <fmip>/api/v1.1/inventory/chassis/cards/3_2?nodeId=<nodeId>`	`{ "slotId”: "3_2" "adminStatus”: "up" }`

   b. Reboot the system.

Request
`GET <fmip>/api/v1.1/clusterConfig/reboot?clusterId=<clusterId>`

   Restarting the card is recommended. When using the reboot request, the script or program needs to pause to give all components time to reinitialize.

6. Create a flow session, specify the buffer count before the match, and enable buffering.

Request	Payload
`POST <fmip>/api/v1.1/saApfProfiles`	`{ "alias": winupdate-sessions, "sessionFields": [{ "pos": 1, "type": "ipv4FiveTuple" }], "timeout": 10, "bidi": False, "buffering": { "enabled": true, "protocol": "tcp", "bufferCountBeforeMatch": 20 } }`

   NOTE: This step sets the number of packets buffered to 20. However, after performing your own investigations, you may want to set the number of packets to a lower value to optimize performance for your particular situation.

7. Configure combined GigaSMART operation.

   The step creates a `gsop` object that references the `saApfProfiles` object created in the *Step 6.*

Request	Payload

`POST <fmip>/api/v1.1/gsops`	`{`
	`  "alias": "winupdate-gsop",`
	`  "gsGroup": "gsgrp1",`
	`  "gsApps": {`
	`  "apf": {`
	`  "enabled": "enabled"},`
	`  "saApf": {`
	`  "saApfProfile": "winupdate-sessions"`
	`  }`
	`  }`
	`}`

8. Create a virtual port and associate it with the gsgroup.

Request	Payload

`POST <fmip>/api/v1.1/vports`	`{`
	`  "alias": "vp1",`
	`  "gsGroup": "gsgrp1"`
	`}`

9. Create a first level map to handle IPv4 traffic.

Request	Payload

`POST <fmip>/api/v1.1/maps`	`{`
	`  "alias" : "ipv4traffic",`
	`  "type" : "firstLevel",`
	`  "subType" : "byRule",`
	`  "srcPorts" : [ "3/2/x5" ],`
	`  "dstPorts" : [ "vp1" ],`
	`  "order" : 2,`
	`  "rules" : {`
	`  "passRules" : [ {`
	`  "ruleId" : 1,`
	`  "bidi" : true,`
	`  "matches" : [ {`
	`  "type" : "ip4Proto",`
	`  "value" : 4`
	`  } ]`
	`  }`
	`}`

10. Create a second level map. The gsrule specifies the traffic to pass.

Request	Payload
POST <fmip>/api/v1.1/maps	{ "alias" : "winupdate_filter", "type" : "secondLevel", "subType" : "byRule", "srcPorts" : ["vp1"], "dstPorts" : ["3/2/x6"], "gsop" : "winpudate-gsop", "gsRules" : { "passRules" : [{ "ruleId" : 1, "matches" : [{ "type" : "pmatch", "value" : "msdownload/update/ software", "matchType" : "regex", "matchOffset" : { "offsetStart" : 0, "protocol" : { "protocol" : "tcp", "pos" : 1 }, "offsetEnd" : 1750 } }] }, { "ruleId" : 2, "matches" : [{ "type" : "pmatch", "value" : "download.windowsupdate.com", "matchType" : "RegEx", "matchOffset" : { "offsetStart" : 0, "protocol" : { "protocol" : "tcp", "pos" : 1 }, "offsetEnd" : 1750 } }] }] } }

11. (Optional) Display the configuration for this example. (Output not shown.)

Request
GET <fmip> api/v1.1/gsgroups/gsgrp1?clusterId=<clusterId> GET <fmip>/api/v1.1/apps/saApfProfiles/winupdate-sessions?clusterId=<clusterId> GET <fmip>/api/v1.1/gsop/winupdate-gsop?clusterId=<clusterId> GET <fmip>/api/v1.1/maps/ipv4traffic?clusterId=<clusterId> GET <fmip>/api/v1.1/maps/winupdate_filter?clusterId=<clusterId>

Filtering Windows 10.0 Upgrade Traffic

This recipe filters traffic that is upgrading the Windows OS to Windows 10.0. For filtering traffic that is upgrading the Windows OS, refer to *Filtering Windows Update Traffic* on page 39.

To set up this recipe, a program or script performs the following steps:

1. Configure the network port.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/ports/3_2_x5?clusterId=<clusterId>`	`{ "portId" : "3/2/x5", "alias" : "", "portType" : "network", "adminStatus" : "up", }`

2. Configure the tool port and enable it.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/ports/3_2_x6?clusterId=<clusterId>`	`{ "portId" : "3/2/x6", "alias" : "", "portType" : "tool", "adminStatus" : "up", }`

3. Configure a GigaSMART group and associate it with GigaSMART engine ports.

Request	Payload
`POST <fmip>/api/v1.1/gsgroups`	`{ "alias" : "gsgrp1" "ports": [ "3/2/e1" ] }`

4. Define the maximum number of sessions. (The ‘2’ indicates a maximum of 2 million sessions.)

Request	Payload
`PUT <fmip>/api/v1.1/gsgroups/gsgrp1/params/saApf`	`{ "bufferSize": 2 }`

5. If this is the first time that an ASF buffer is being configured, restart the GigaSMART engine. A script or program can restart the engine by doing either of the following:

   - Send two PATCH requests. The fist request change the card's status to `down`. The second request changes the status to `up`, which restarts the engine.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/chassis/cards/3_2?nodeId=<nodeId>`	`{ "slotId”: "3_2", "adminStatus": "down" }`
`PATCH <fmip>/api/v1.1/inventory/chassis/cards/3_2?nodeId=<nodeId>`	`{ "slotId”: "3_2" "adminStatus”: "up" }`

   - Reboot the system.

Request
`GET <fmip>/api/v1.1/clusterConfig/reboot?clusterId=<clusterId>`

   Restarting the card is recommended. When using the reboot request, the script or program needs to pause to give all components time to reinitialize.

6. Create a flow session, specify the buffer count before the match, and enable buffering.

Request	Payload
`POST <fmip>/api/v1.1/saApfProfiles`	`{ "alias": winupgrade-sessions, "sessionFields": [{ "pos": 1, "type": "ipv4FiveTuple" }], "timeout": 10, "bidi": False, "buffering": { "enabled": true, "protocol": "tcp", "bufferCountBeforeMatch": 20 } }`

   NOTE: This step sets the number of packets buffered to 20. However, after performing your own investigations, you may want to set the number of packets to a lower value to optimize performance for your particular situation.

7. Configure combined GigaSMART operation.

   The step creates a `gsop` object that references the `saApfProfiles` object created in *Step 6.*

Request	Payload

`POST <fmip>/api/v1.1/gsops`	`{`
	`  "alias": "winupgrade-gsop",`
	`  "gsGroup": "gsgrp1",`
	`  "gsApps": {`
	`  "apf": {`
	`  "enabled": "enabled"},`
	`  "saApf": {`
	`  "saApfProfile": "winupgrade-sessions"`
	`  }`
	`  }`
	`}`

8. Create a virtual port and associate it with the gsgroup.

Request	Payload

`POST <fmip>/api/v1.1/vports`	`{`
	`  "alias": "vp1",`
	`  "gsGroup": "gsgrp1"`
	`}`

9. Create a first level map to handle IPv4 traffic.

Request	Payload

`POST <fmip>/api/v1.1/maps`	`{`
	`  "alias" : "ipv4traffic",`
	`  "type" : "firstLevel",`
	`  "subType" : "byRule",`
	`  "srcPorts" : [ "3/2/x5" ],`
	`  "dstPorts" : [ "vp1" ],`
	`  "rules" : {`
	`  "passRules" : [ {`
	`  "ruleId" : 1,`
	`  "bidi" : true,`
	`  "matches" : [ {`
	`  "type" : "ip4Proto",`
	`  "value" : 4`
	`  } ]`
	`  }`
	`}`

10. Create a second level map. The gsrule specifies the traffic to pass.

Request	Payload
POST <fmip>/api/v1.1/maps	{ "alias" : "winupgrade_filter", "type" : "secondLevel", "subType" : "byRule", "srcPorts" : ["vp1"], "dstPorts" : ["3/2/x6"], "gsop" : "winupgrade-gsop", "gsRules" : { "passRules" : [{ "ruleId" : 1, "matches" : [{ "type" : "pmatch", "value" : "update.*windows10.0", "matchType" : "regex", "matchOffset" : { "offsetStart" : 0, "protocol" : { "protocol" : "tcp", "pos" : 1 }, "offsetEnd" : 1750 } }] }, { "ruleId" : 2, "matches" : [{ "type" : "pmatch", "value" : ".esd", "matchType" : "string", "matchOffset" : { "offsetStart" : 0, "protocol" : { "protocol" : "tcp", "pos" : 1 }, "offsetEnd" : 1750 } }] }] } }

11. (Optional) Display the configuration for this example. (Output not shown.)

Request
GET <fmip> api/v1.1/gsgroups/gsgrp1?clusterId=<clusterId> GET <fmip>/api/v1.1/apps/saApfProfiles/winupgrade-sessions?clusterId=<clusterId> GET <fmip>/api/v1.1/gsop/winupgrade-gsop?clusterId=<clusterId> GET <fmip>/api/v1.1/maps/ipv4traffic?clusterId=<clusterId> GET <fmip>/api/v1.1/maps/winupgrade_filter?clusterId=<clusterId>

Filtering Emails with Attachments

This recipe passes all email traffic containing attachments.

To set up this recipe, a script or program performs the following steps:

1. Configure the network port.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/ports/3_2_x5?clusterId=<clusterId>`	`{ "portId" : "3/2/x5", "alias" : "", "portType" : "network", "adminStatus" : "up", }`

2. Configure the tool port and enable it.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/ports/3_2_x6?clusterId=<clusterId>`	`{ "portId" : "3/2/x6", "alias" : "", "portType" : "tool", "adminStatus" : "up", }`

3. Configure a GigaSMART group and associate it with GigaSMART engine ports.

Request	Payload
`POST <fmip>/api/v1.1/gsgroups`	`{ "alias" : "gsgrp1" "ports": [ "3/2/e1" ] }`

4. Define the maximum number of sessions. (The ‘2’ indicates a maximum of 2 million sessions.)

Request	Payload
`PUT <fmip>/api/v1.1/gsgroups/gsgrp1/params/saApf`	`{ "bufferSize": 2 }`

5. If this is the first time that an ASF buffer is being configured, restart the GigaSMART engine. A script or program can restart the engine by doing either of the following:

   - Send two PATCH requests. The fist request change the card's status to `down`. The second request changes the status to `up`, which restarts the engine.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/chassis/cards/3_2?nodeId=<nodeId>`	`{ "slotId”: "3_2", "adminStatus": "down" }`
`PATCH <fmip>/api/v1.1/inventory/chassis/cards/3_2?nodeId=<nodeId>`	`{ "slotId”: "3_2" "adminStatus”: "up" }`

   - Reboot the system.

Request
`GET <fmip>/api/v1.1/clusterConfig/reboot?clusterId=<clusterId>`

   Restarting the card is recommended. When using the reboot request, the script or program needs to pause to give all components time to reinitialize.

6. Create a flow session, specify the buffer count before the match, and enable buffering.

Request	Payload
`POST <fmip>/api/v1.1/saApfProfiles`	`{ "alias": email-sessions, "sessionFields": [{ "pos": 1, "type": "ipv4FiveTuple" }], "timeout": 10, "bidi": False, "buffering": { "enabled": true, "protocol": "tcp", "bufferCountBeforeMatch": 20 } }`

   NOTE: This step sets the number of packets buffered to 20. However, after performing your own investigations, you may want to set the number of packets to a lower value to optimize performance for your particular situation.

7. Configure combined GigaSMART operation.

   This step creates a `gsop` object that references the `saApfProfiles` object created in the *Step 6.*

Request	Payload

`POST <fmip>/api/v1.1/gsops`	`{`
	`  "alias": "email-gsop",`
	`  "gsGroup": "gsgrp1",`
	`  "gsApps": {`
	`  "apf": {`
	`  "enabled": "enabled"},`
	`  "saApf": {`
	`  "saApfProfile": "email-sessions"`
	`  }`
	`  }`
	`}`

8. Create a virtual port and associate it with the gsgroup.

Request	Payload

`POST <fmip>/api/v1.1/vports`	`{`
	`  "alias": "vp1",`
	`  "gsGroup": "gsgrp1"`
	`}`

9. Create a first level map to filter SMTP traffic.

Request	Payload

`POST <fmip>/api/v1.1/maps`	`{`
	`  "alias" : "smtptraffic",`
	`  "type" : "firstLevel",`
	`  "subType" : "byRule",`
	`  "dstPorts" : [ "vp1" ],`
	`  "rules" : {`
	`  "passRules" : [ {`
	`  "ruleId" : 1,`
	`  "bidi" : true,`
	`  "matches" : [ {`
	`  "type" : "portSrc",`
	`  "value" : 25`
	`  } ]`
	`  } ]`
	`  }`
	`}`

10. Create a second level map to decrypt emails with attachments. The gsrule specifies the traffic to drop, using keywords. Buffered packets and all subsequent packets will be passed.

Request	Payload
`POST <fmip>/api/v1.1/maps?clusterId=<clusterId>`	`{ "alias" : "email_filter", "type" : "secondLevel", "subType" : "byRule", "srcPorts" : ["vp1"], "dstPorts" : ["3/2/x6"], "gsop" : "email-gsop", "gsRules" : { "passRules" : [{ "ruleId" : 1, "matches" : [{ "type" : "pmatch", "value" : "Content-Disposition", "matchType" : "regex", "matchOffset" : { "offsetStart" : 0, "protocol" : { "protocol" : "tcp", "pos" : 1 }, "offsetEnd" : 1750 } }] }] }`

11. (Optional) Display the configuration for this example. (Output not shown.)

Request

`GET <fmip> api/v1.1/gsgroups/gsgrp1?clusterId=<clusterId>`
`GET <fmip>/api/v1.1/apps/saApfProfiles/email-sessions?clusterId=<clusterId>`
`GET <fmip>/api/v1.1/gsop/email-gsop?clusterId=<clusterId>`
`GET <fmip>/api/v1.1/maps/v4filter?clusterId=<clusterId>`
`GET <fmip>/api/v1.1/maps/email_filter?clusterId=<clusterId>`

Filtering Traffic with SSL Decryption

In some cases it is useful to drop high-bandwidth encrypted traffic and alleviate the tools from the demands of decrypting and filtering. In this scenario, a corporation is looking to protect its HTTPS server, which is defined as follows:

- The server is inside the organization.
- There is access to the server's SSL private key.
- Browser client requests can be users inside or outside the organization.

Some example use cases are: Application Performance Monitoring, Data Loss Prevention (Detection), Web Threat Detection, Security Analytics, and Forensic Traffic Capture.

The following recipe shows how to apply APF and ASF to encrypted traffic. This is accomplished with an SSL map that decrypts traffic using a private key then sends the traffic to a hybrid port. (In the recipe, traffic from a Confluence server is used as an example.) The hybrid port is used to internally loop the traffic back to a virtual port where APF and ASF filtering is performed. This is done because SSL decryption cannot be combined with APF or ASF, so separate maps are needed for each.

To set up this recipe, a script or program performs the following steps:

1. Configure the network port.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/ports/3_2_x5?clusterId=<clusterId>`	`{ "portId" : "1/3/x1", "alias" : "", "portType" : "network", "adminStatus" : "up", }`

2. Configure the hybrid port.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/ports/1_3_x5?clusterId=<clusterId>`	`{ "portId" : "1/3/x5", "alias" : "", "portType" : "hybrid", "adminStatus" : "up", }`

3. Install the private key and create an SSL service.

Request	Payload
`POST <fmip>/api/v1.1/apps/ssl/keys/file?clusterId=<clusterId>`	`{ "alias : "confluence_key", "type : "privateKey", "key” : “url sftp:<path_to_private_key>"`

Request	Payload
POST <fmip>/api/v1.1/apps/ssl/endpoints?clusterId=<clusterId>	{ "alias : "confluence", "address" : "10.80.1.75" }

4. Configure a GigaSMART group and associate it with GigaSMART engine ports..

Request	Payload
POST <fmip>/api/v1.1/gsgroups?clusterId=<clusterId>	{ "alias" : "gs1" "ports": ["3/2/e1"] }

5. Create a GigaSMART group for SSL decryption.

Request	Payload
POST <fmip>/api/v1.1/gsgroups?clusterId=<clusterId>	{ "gsGroup": "gs1", "alias": "ssl", "gsApps": { "sslDecrypt": { "outPort": 0, "inPort": 0}, "dedup": "enabled": "enabled" } } }
POST <fmip>/api/v1.1/keyMaps?clusterId=<clusterId>	{ "alias": "mappings" : [{ "endpointAlias" : "confluence", "keyAlias" : "confluence_key",] } }

6. Create a first level map to filter all traffic between a single host and the Confluence server. In this example, port 1/3/x1 is a network port and the source of the Confluence traffic. The decrypted traffic from port 1/3/x1 is sent to hybrid port 1/3/x5.

Request	Payload
POST <fmip>/api/v1.1/ maps?clusterId=<clusterId>	{ "alias" : "ssl", "type" : "regular", "subType" : "byRule", "srcPorts" : ["**1/3/x1**"], "dstPorts" : ["**1/3/x5**"], "gsop" : "ssl", "order" : 1, "rules" : { "passRules" : [{ "ruleId" : 1, "bidi" : true, "matches" : [{ "type" : "ip4Dst", "value" : "10.50.22.21", "netMask" : "255.255.255.255" }, { "type" : "ip4Src", "value" : "10.80.1.75", "netMask" : "255.255.255.255" }] }] }, "ruleMatching" : "normal" }

7. Create a session field group.

Request	Payload
POST <fmip>/api/v1.1/ saApfProfiles?clusterId=<clusterId>	{ "alias" : "video_sessions", "sessionFields" : [{ "pos" : 1, "type" : "ipv4FiveTuple" }], "timeout" : 15, "npacket" : 0, "bidi" : true, "buffering" : { "enabled" : false, "protocol" : "tcp", "bufferCountBeforeMatch" : 3 } }

8. Create a GigaSMART operation with non-buffered APF

   This step creates a `gsop` object that references the `saApfProfiles` object created in *Step 7.*

Request	Payload
`POST <fmip>/api/v1.1/gsops?clusterId=<clusterId>`	`{ "alias" : "nb_video_asf", "gsGroup" : "gs1", "gsApps" : { "apf" : { "enabled" : "enabled" }, "saApf" : { "saApfProfile" : "video_sessions" } } }`

9. Define the maximum number of sessions. (The '2' indicates a maximum of 2 million sessions.)

Request	Payload
`PUT <fmip>/api/v1.1/gsgroups/gs1/params/saApf?clusterId=<clusterId>`	`{ "bufferSize": 2 }`

10. If this is the first time that an ASF buffer is being configured, restart the GigaSMART engine. A script or program can restart the engine by doing either of the following:

    - Send two PATCH requests. The fist request change the card's status to `down`. The second request changes the status to `up`, which restarts the engine.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/chassis/cards/3_2?nodeId=<nodeId>`	`{ "slotId": "3_2", "adminStatus": "down" }`
`PATCH <fmip>/api/v1.1/inventory/chassis/cards/3_2?nodeId=<nodeId>`	`{ "slotId": "3_2" "adminStatus": "up" }`

    - Reboot the system.

Request
`GET <fmip>/api/v1.1/clusterConfig/reboot?clusterId=<clusterId>`

    Restarting the card is recommended. When using the reboot request, the script or program needs to pause to give all components time to reinitialize.

11. Create a virtual port and associate it with the gsgroup.

Request	Payload
`POST <fmip>/api/v1.1/vports?clusterId=<clusterId>`	`{ "alias": "vp1", "gsGroup": "gs1" }`

12. Create an ingress first level map.

Request	Payload
`POST <fmip>/api/v1.1/maps?clusterId=<clusterId>`	`{ "alias" : "video1", "type" : "firstLevel", "subType" : "byRule", "srcPorts" : [ "1/3/x5" ], "dstPorts" : [ "vp1" ], "order" : 2, "rules" : { "passRules" : [ { "ruleId" : 1, "bidi" : true, "matches" : [ { "type" : "portDst", "value" : 80 } ] } ] } }`

13. Create an egress second level map.

Request	Payload
`POST <fmip>/api/v1.1/maps?clusterId=<clusterId>`	`{ "alias" : "video2", type" : "secondLevel", "subType" : "byRule", "srcPorts" : ["vp1"], "dstPorts" : ["1/3/x6"], "gsop" : "nb_video_asf", "gsRules" : { "passRules" : [{ "ruleId" : 1, "matches" : [{ "type" : "pmatch", "value" : "video/mp4", "matchType" : "string", "matchOffset" : { "offsetStart" : 0, "protocol" : { "protocol" : "tcp", "pos" : 1 }, "offsetEnd" : 1500 } }] }] }`

14. (Optional) Display the configuration for this example. (Output not shown.)

Request

```
GET <fmip> api/v1.1/gsgroups/gs1?clusterId=<clusterId>
GET <fmip>/api/v1.1/apps/saApfProfiles/video_sessions?clusterId=<clusterId>
GET <fmip>/api/v1.1/gsop/nb_video_asf?clusterId=<clusterId>
GET <fmip>/api/v1.1/maps/ssl?clusterId=<clusterId>
GET <fmip>/api/v1.1/maps/video1?clusterId=<clusterId>
GET <fmip>/api/v1.1/maps/video2?clusterId=<clusterId>
```

Filtering VOIP Traffic (SIP-Based)

To filter and drop Voice-Over-IP (VOIP) traffic, this recipe first filters UDP packets that use a specific port range used by the VOIP system, then uses a regular expression pattern to match a range of valid bit values with the first 2 bytes of the RTP header to filter for RTP/RTCP packets, which contains the audio data.

To filter for SIP packets (call setup) and RTP/RTCP packets (audio data), perform the following steps:

1. Configure the network port.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/ports/3_2_x5?clusterId=<clusterId>`	`{ "portId" : "3/2/x5", "alias" : "", "portType" : "network", "adminStatus" : "up", }`

2. Configure the tool port and enable it.

Request	Payload
`PATCH <fmip>/api/v1.1/inventory/ports/3_2_x6?clusterId=<clusterId>`	`{ "portId" : "3/2/x6", "alias" : "", "portType" : "tool", "adminStatus" : "up", }`

3. Create a session field group for SIP packets.

Request	Payload
`POST <fmip>/api/v1.1/saApfProfiles?clusterId=<clusterId>`	`"saApfProfile" : { "alias" : "voip_sessions", "sessionFields" : [ { "pos" : 1, "type" : "ipv4FiveTuple" } ], "timeout" : 15, "bidi" : true, "buffering" : { "enabled" : true, "protocol" : "udp", "bufferCountBeforeMatch" : 10 } } }`

4. Configure a GigaSMART group and associate it with GigaSMART engine ports.

Request	Payload

POST <fmip>/api/v1.1/
gsgroups?clusterId=<clusterId>
	{
	"alias" : "gsgrp1"
	"ports": [
	"3/2/e1"
	]
	}

5. Create a GigaSMART operation with non-buffered APF.

Request	Payload

POST <fmip>api/v1.1/
gsops?clusterId=<clusterId>
	{
	"alias" : "voip-gsop",
	"gsGroup" : "gsgrp1",
	"gsApps" : {
	"apf" : {
	"enabled" : "enabled"
	},
	"saApf" : {
	"saApfProfile" : "voip_sessions"
	}
	}
	}

6. Create a virtual port and associate it with the gsgroup.

Request	Payload

POST <fmip>/api/v1.1/
vports?clusterId=<clusterId>
	{
	"alias": "vp1",
	"gsGroup": "gsgrp1"
	}

7. Create a first level map for ingress packets.

Request	Payload

POST <fmip>/api/v1.1/
maps?clusterId=<clusterId>
	<pre>{
	  "alias" : "voip1",
	  "type" : "firstLevel",
	  "subType" : "byRule",
	  "srcPorts" : ["3/2/x5"],
	  "dstPorts" : ["vp1"],
	  "order" : 1,
	  "rules" : {
	  "passRules" : [{
	  "ruleId" : 1,
	  "bidi" : true,
	  "matches" : [{
	  "type" : "portSrc",
	  "value" : 5060,
	  "valueMax" : 5061
	  }, {
	  "type" : "ip4Proto",
	  "value" : 17
	  }]
	  }, {
	  "ruleId" : 2,
	  "bidi" : true,
	  "matches" : [{
	  "type" : "portSrc",
	  "value" : 16384,
	  "valueMax" : 32767
	  }, {
	  "type" : "ip4Proto",
	  "value" : 17
	  }]
	  }]
	  }
	}</pre>

8. Create a second level map for egress traffic.

Request	Payload

POST <fmip>/api/v1.1/
maps?clusterId=<clusterId>
	<pre>{
	"alias" : "voip2",
	"type" : "secondLevel",
	"subType" : "byRule",
	"srcPorts" : ["vp1"],
	"dstPorts" : ["3/2/x6"],
	"gsop" : "voip-gsop",
	"gsRules" : {
	"passRules" : [{
	"ruleId" : 1,
	"matches" : [{
	"type" : "pmatch",
	"value" : "SIP/2.0",
	"matchType" : "string",
	"matchOffset" : {
	"offsetStart" : 0,
	"protocol" : {
	"protocol" : "udp",
	"pos" : 1
	},
	"offsetEnd" : 200
	}
	}]
	}, {
	"ruleId" : 2,
	"matches" : [{
	"type" : "pmatch",
	"value" :
	"[\\x80-\\xBF][\\x00|\\x03-\\x12|\\x19|\\x1A|\\
	x1C|\\x1F-\\x22|\\x60-\\x7F|\\x80|\\x83-\\x92|\
	\x99|\\x9A|\\x9C|\\x9F-\\xA2|\\xC0-\\xFF]",
	"matchType" : "regex",
	"matchOffset" : {
	"offsetStart" : 0,
	"protocol" : {
	"protocol" : "udp",
	"pos" : 1
	},
	"offsetEnd" : 2
	}
	}]
	}]
	}
	}</pre>

9. (Optional) Display the configuration for this example. (Output not shown.)

Request
GET <fmip> api/v1.1/gsgroups/gs1?clusterId=<clusterId> GET <fmip>/api/v1.1/apps/saApfProfiles/voip_sessions?clusterId=<clusterId> GET <fmip>/api/v1.1/gsop/voip-gsop?clusterId=<clusterId> GET <fmip>/api/v1.1/maps/voip1?clusterId=<clusterId> GET <fmip>/api/v1.1/maps/voip2?clusterId=<clusterId>

Appendix A

CLI Recipe Scripts

This appendix provides a list of the commands for each recipe described in Chapter 2, *Application Session Filtering Recipes*. They are provided in a format that is easy to copy into a text file. After editing the port IDs so they match your particular configuration, you can upload the file to a node and run it as a script.

To upload the text file and apply the commands, do the following:

1. Use the following command to upload the file to the node:

   (config) # configuration text fetch ftfp:<ip address>/<text file>

   For example, to fetch the commands for dropping Neflix traffic in a file named AsfNetflixFilter.txt from address 10.40.21.124, the command is:

   (config) # configuration text fetch tftp://10.40.21.124/ASFNetflixFilterPart2.txt

2. Use the following to apply the commands:

   (config) # configuration text file <text file> apply

   For example:

   (config) # configuration text file ASFNetflixFilter.txt apply

NOTE: The first time you configure an ASF buffer, you need to restart the GigaSMART engine using the following commands. The first command brings down the engine. The second command brings it back up.

card slot [slot ID] down
no card slot [slot ID] down

Filtering Netflix, YouTube, or Facebook Traffic

This section provides the commands for dropping or passing Netflix, YouTube, or Facebook traffic.

Dropping Traffic

Use these scripts to drop traffic.

Dropping Netflix Traffic

```
## ASF Configuration to drop Netflix traffic

## Network port configuration example
## port 1/1/g1..g4 params admin enable
port <port-list> params admin enable

## Tool Port Configuration example
## port 1/1/g5 type tool
## port 1/1/g5 params admin enable
port <port-id> type tool
port <port-id> params admin enable

## GS group configuration example (GS goup is created and associated to the GS engine)
gsgroup alias gsgrp1 port-list 1/1/e1
gsgroup alias gsgrp1 port-list <port-list>

## GS params configuration example (Buffer limits are set for the ASF application)
gsparams gsgroup gsgrp1 resource buffer-asf 2

## Reload GS Card example
## card slot 1/1 down
## no card slot 1/1 down
card slot <box ID>/slot ID> down
no card slot <box ID>/<slot ID> down

## APF configuration example (Enabling / disabling buffering for sessions & session creating criteria)
apps asf alias netflix-sessions
   bi-directional enable
   buffer enable
   buffer-count-before-match 20
   sess-field add ipv4-5tuple outer
  exit

## Gsop configurations (GigaSMART operation for ASF function)
gsop alias netflix-gsop apf set asf netflix-sessions port-list gsgrp1

## Vport configurations (Virtual port configuration to feed the egress traffic from first level maps)
vport alias vp1 gsgroup gsgrp1

## Traffic map connection configuration example (First level map to filter Web Traffic)
## map alias webtraffic
##    rule add pass portdst 80 bidir
##    rule add pass portdst 443 bidir
##    rule add pass portdst 8080 bidir
##    to vp1
##    from 1/1/g1..g4
##    exit
map alias webtraffic
   rule add pass portdst 80 bidir
   rule add pass portdst 443 bidir
   rule add pass portdst 8080 bidir
   to vp1
```

```
   from <port-list>
   exit

## ASF second level map configuration to drop Netflix traffic

##Second Level map to filter Netflix traffic example (Second level map to filter Netflix sessions)
## map alias netflix_filter
##    use gsop netflix-gsop
##    gsrule add drop pmatch protocol tcp pos 1 RegEx netflix|nflxvideo|nflximg|Netflix|nflxext|nflxsearch
0..1000
##    gsrule add drop pmatch protocol tcp pos 1 string octet-stream 0..1000
##    to 1/1/g5
##    from vp1
##    exit
map alias netflix_filter
   use gsop netflix-gsop
   gsrule add drop pmatch protocol tcp pos 1 RegEx netflix|nflxvideo|nflximg|Netflix|nflxext|nflxsearch 0..1000
   gsrule add drop pmatch protocol tcp pos 1 string octet-stream 0..1000
   to <port-id>
   from vp1
   exit

## Pass rest of the traffic to the collector / tools example
## map-scollector alias collector
##    from vp1
##    collector 1/1/g5
##    exit
map-scollector alias collector
   from vp1
   collector <port-id>
   exit
```

Dropping YouTube Traffic

```
## ASF configuration to drop YouTube traffic

## Network port configuration example
## port 1/1/g1..g4 params admin enable
port <port-list> params admin enable

## Tool port configuration example
## port 1/1/g5 type tool
## port 1/1/g5 params admin enable
port <port-id> type tool
port <port-id> params admin enable

## GS group configuration example (GS group is created and associated to the GS engine)
## gsgroup alias gsgrp1 port-list 1/1/e1
gsgroup alias gsgrp1 port-list <port-list>

## GS params configuration example (Buffer limits are set for the ASF application)
gsparams gsgroup gsgrp1 resource buffer-asf 2

## Reload GS Card
# card slot 1/1 down
# no card slot 1/1 down
card slot <box ID>/<slot ID> down
no card slot <box ID>/<slot ID> down

## APF configuration exmaple (Enabling / disabling buffering for sessions & session creating criteria)
apps asf alias youtube-sessions
   bi-directional enable
   buffer enable
   buffer-count-before-match 20
   sess-field add ipv4-5tuple outer
   exit
```

```
## Gsop configurations (GigaSMART operation for ASF function)
gsop alias youtube-gsop apf set asf youtube-sessions port-list gsgrp1

## Vport configuration (Virtual port configuration to feed the egress traffic from first level maps)
vport alias vp1 gsgroup gsgrp1

## Traffic map connection configuration example (First level map to filter Web Traffic)
##map alias webtraffic
##    rule add pass portdst 80 bidir
##    rule add pass portdst 443 bidir
##    to vp1
##    from 1/1/g1..g4
##    exit
map alias webtraffic
   rule add pass portdst 80 bidir
   rule add pass portdst 443 bidir
   to vp1
   from <port-list>
   exit

## ASF Second Level Map Configuration to Filter Youtube Traffic example
##map alias youtube_filter
##    use gsop youtube-gsop
##    gsrule add drop pmatch protocol tcp pos 1 RegEx youtube|ytimg|yt3.ggpht|tubeMogul|tmogul|youtu
0..1000
##    gsrule add drop pmatch protocol tcp pos 1 string "Content-Type: video" 0..1000
##    gsrule add drop pmatch protocol tcp pos 1 string "octet-stream" 0..1750
##    to 1/1/g5
##    from vp1
##    exit
map alias youtube_filter
   use gsop youtube-gsop
   gsrule add drop pmatch protocol tcp pos 1 RegEx youtube|ytimg|yt3.ggpht|tubeMogul|tmogul|youtu 0..1000
   gsrule add drop pmatch protocol tcp pos 1 string "Content-Type: video" 0..1000
   gsrule add drop pmatch protocol tcp pos 1 string "octet-stream" 0..1750
   to <port-id>
   from vp1
   exit

## Pass rest of the traffic to the collector / tools example
##map-scollector alias collector
##    from vp1
##    collector 1/1/g5
##    exit
map-scollector alias collector
   from vp1
   collector <port-id>
   exit
```

Dropping Facebook Traffic

```
## ASF configuration to drop Facebook traffic

## Network port configuration example
## port 1/1/g1..g4 params admin enable
port <port-list> params admin enable

## Tool Port Configuration example
## port 1/1/g5 type tool
## port 1/1/g5 params admin enable
port <port-id> type tool
port <port-id> params admin enable

## GS group configuration example (GS group is created and associated to the GS engine)
## gsgroup alias gsgrp1 port-list 1/1/e1
gsgroup alias gsgrp1 port-list <port-list>
```

```
## GS params configuration (Buffer limits are set for the ASF application)
gsparams gsgroup gsgrp1 resource buffer-asf 2

## Reload GS Card example
# card slot 1/1 down
# no card slot 1/1 down
card slot <box ID>/<slot ID> down
no card slot <box ID>/<slot ID> down

## APF configuration example (Enabling / disabling buffering for sessions & session creating criteria)
apps asf alias fb-sessions
   bi-directional enable
   buffer enable
   buffer-count-before-match 20
   sess-field add ipv4-5tuple outer
exit

## Gsop configuration example (Gigasmart opertation for ASF function)
gsop alias fb-gsop apf set asf fb-sessions port-list gsgrp1

## Vport configuration example (Virtual port configuration to feed the egress traffic from first level maps)
vport alias vp1 gsgroup gsgrp1

## Traffic map connection configuration example (First level map to filter Web Traffic)
##map alias webtraffic
##    rule add pass portdst 80 bidir
##   rule add pass portdst 443 bidir
##    to vp1
##    from 1/1/g1..g4
##    exit
map alias webtraffic
   rule add pass portdst 80 bidir
   rule add pass portdst 443 bidir
   to vp1
   from <port-list>
   exit

## ASF Second level map configuration to filter Facebook traffic example
##map alias fb_filter
##    use gsop fb-gsop
##    gsrule add drop pmatch protocol tcp pos 1 RegEx
^fdcdn.*\.akamaihd\.net|fbstatic.*\.akamaihd\.net|fbexternal.*\.akamaihd\.net|.*\.facebook\.com|.*\.fbcdn\.net
0..1750
##    to 1/1/g5
##    from vp1
##exit
map alias fb_filter
   use gsop fb-gsop
   gsrule add drop pmatch protocol tcp pos 1 RegEx
^fdcdn.*\.akamaihd\.net|fbstatic.*\.akamaihd\.net|fbexternal.*\.akamaihd\.net|.*\.facebook\.com|.*\.fbcdn\.net
0..1750
   to <port-id>
   from vp1
exit

## Pass rest of the traffic to the collector / tools example
##map-scollector alias collector
##    from vp1
##    collector 1/1/g5
##    exit
map-scollector alias collector
   from vp1
   collector <port-id>
   exit
```

Passing Traffic

Use these scripts to pass traffic.

Passing Netflix Traffic

```
## ASF configuration to pass Netflix traffic to Tools / Collector example

## Network Port Configuration example
## port 1/1/g1..g4 params admin enable
port <port-list> params admin enable

## Tool Port Configuration example
## port 1/1/g5 type tool
## port 1/1/g5 params admin enable
port <port-id> type tool
port <port-id> params admin enable

## Gsgroup configuration example (Group is created and associated to the GS engine)
## gsgroup alias gsgrp1 port-list 1/1/e1
gsgroup alias <alias> port-list <port-list>

## Gs params configuration example (Buffer limits are set for the ASF application)
gsparams gsgroup gsgrp1 resource buffer-asf 2

## Reload GS Card example
# card slot 1/1 down
# no card slot 1/1 down
card slot <box ID>/<slot ID> down
no card slot <box ID>/<slot ID> down

## SAPF configuration example (Enabling / disabling buffering for sessions & session creating criteria)
apps asf alias netflix-sessions
   bi-directional enable
   buffer enable
   buffer-count-before-match 20
   sess-field add ipv4-5tuple outer
   exit

## Gsop configuration example (Gigasmart opertation for ASF function)
gsop alias netflix-gsop apf set asf netflix-sessions port-list gsgrp1

## Vport configuration example (Virtual port configuration to feed the egress traffic from first level maps)
vport alias vp1 gsgroup gsgrp1

## Traffic map connection configuration example (First level map to filter Web Traffic)
##map alias webtraffic
##    rule add pass portdst 80 bidir
##    rule add pass portdst 443 bidir
##    to vp1
##    from 1/1/g1..g4
##    exit
map alias webtraffic
   rule add pass portdst 80 bidir
   rule add pass portdst 443 bidir
   to vp1
   from <port-list>
   exit

## ASF second level map configuration to filter Netflix traffic example
##map alias netflix_filter
##    use gsop netflix-gsop
##    gsrule add pass pmatch protocol tcp pos 1 RegEx netflix|nflxvideo|nflximg|Netflix|nflxext|nflxsearch
0..1000
##    gsrule add pass pmatch protocol tcp pos 1 string octet-stream 0..1000
##    to 1/1/g5
##    from vp1
```

```
##    exit
map alias netflix_filter
   use gsop netflix-gsop
   gsrule add pass pmatch protocol tcp pos 1 RegEx netflix|nflxvideo|nflximg|Netflix|nflxext|nflxsearch 0..1000
   gsrule add pass pmatch protocol tcp pos 1 string octet-stream 0..1000
   to <port-id>
   from vp1
   exit
```

Passing YouTube Traffic

```
## ASF configuration to pass YouTube traffic

## Network Port Configuration example
## port 1/1/g1..g4 params admin enable
port <port-list> params admin enable

## Tool Port Configuration example
## port 1/1/g5 type tool
## port 1/1/g5 params admin enable
port <port-id> type tool
port <port-id> params admin enable

## Gsgroup configuration example (Group is created and associated to the GS engine)
## gsgroup alias gsgrp1 port-list 1/1/e1
gsgroup alias <alias> port-list <port-list>

## Gs params configuration example (Buffer limits are set for the ASF application)
gsparams gsgroup gsgrp1 resource buffer-asf 2

## Reload GS Card example
# card slot 1/1 down
# no card slot 1/1 down
card slot <box ID>/<slot ID> down
no card slot <box ID>/<slot ID> down

## SAPF configuration example (Enabling / disabling buffering for sessions & session creating criteria)
apps asf alias youtube-sessions
   bi-directional enable
   buffer enable
   buffer-count-before-match 20
   sess-field add ipv4-5tuple outer
   exit

## Gsop configuration example (GigaSMART operation for ASF function)
gsop alias youtube-gsop apf set asf youtube-sessions port-list gsgrp1

## Vport configuration example (Virtual port configuration to feed the egress traffic from first level maps)
vport alias vp1 gsgroup gsgrp1

## Traffic map connection configuration example (First level map to filter Web traffic)
##map alias webtraffic
##    rule add pass portdst 80 bidir
##    rule add pass portdst 443 bidir
##    to vp1
##    from 1/1/g1..g4
##    exit
map alias webtraffic
   rule add pass portdst 80 bidir
   rule add pass portdst 443 bidir
   to vp1
   from <port-list>
   exit

## ASF Second Level Map Configuration to Filter Youtube Traffic example
##map alias youtube_filter
##    use gsop youtube-gsop
##    gsrule add pass pmatch protocol tcp pos 1 RegEx youtube|ytimg|yt3.ggpht|tubeMogul|tmogul|youtu
```

```
0..1000
##    gsrule add pass pmatch protocol tcp pos 1 string "Content-Type: video" 0..1000
##    gsrule add pass pmatch protocol tcp pos 1 string "octet-stream" 0..1750
##    to 1/1/g5
##    from vp1
##    exit
map alias youtube_filter
   use gsop youtube-gsop
   gsrule add pass pmatch protocol tcp pos 1 RegEx youtube|ytimg|yt3.ggpht|tubeMogul|tmogul|youtu 0..1000
   gsrule add pass pmatch protocol tcp pos 1 string "Content-Type: video" 0..1000
   gsrule add pass pmatch protocol tcp pos 1 string "octet-stream" 0..1750
   to <port-id>
   from vp1
   exit
```

Passing Facebook Traffic

```
## ASF Configuration to Pass Facebook Traffic

### Network Port Configuration example
## port 1/1/g1..g4 params admin enable
port <port-list> params admin enable

## Tool Port Configuration example
## port 1/1/g5 type tool
## port 1/1/g5 params admin enable
port <port-id> type tool
port <port-id> params admin enable

## GS group configuration example (GS group is created and associated to the GS engine)
## gsgroup alias gsgrp1 port-list 1/1/e1
gsgroup alias <alias> port-list <port-list>

## GS params configuration example (Buffer limits are set for the ASF application)
gsparams gsgroup gsgrp1 resource buffer-asf 2

## Reload GS Card example
# card slot 1/1 down
# no card slot 1/1 down
card slot <box ID>/<slot ID> down
no card slot <box ID>/<slot ID> down

## APF configuration example (Enabling / disabling buffering for sessions & session creating criteria)
apps asf alias fb-sessions
   bi-directional enable
   buffer enable
   buffer-count-before-match 20
   sess-field add ipv4-5tuple outer
   exit

## Gsop configuration example (GigaSMART operation for ASF function)
gsop alias fb-gsop apf set asf fb-sessions port-list gsgrp1

## Vport configuration example (Virtual port configuration to feed the egress traffic from first level maps)
vport alias vp1 gsgroup gsgrp1

## Traffic map connection configuration example (First level map to filter Web Traffic)
##map alias webtraffic
##    rule add pass portdst 80 bidir
##    rule add pass portdst 443 bidir
##    to vp1
##    from 1/1/g1..g4
##    exit
map alias webtraffic
   rule add pass portdst 80 bidir
   rule add pass portdst 443 bidir
   to vp1
```

```
   from <port-list>
   exit

## ASF second level map configuration tofFilter Facebook traffic example
##map alias fb_filter
##    use gsop fb-gsop
##    gsrule add pass pmatch protocol tcp pos 1 RegEx
^fdcdn.*\.akamaihd\.net|fbstatic.*\.akamaihd\.net|fbexternal.*\.akamaihd\.net|.*\.facebook\.com|.*\.fbcdn\.net
0..1750
##    to 1/1/g5
##    from vp1
#    exit
map alias fb_filter
   use gsop fb-gsop
   gsrule add pass pmatch protocol tcp pos 1 RegEx
^fdcdn.*\.akamaihd\.net|fbstatic.*\.akamaihd\.net|fbexternal.*\.akamaihd\.net|.*\.facebook\.com|.*\.fbcdn\.net
0..1750
   to <port-id>
   from vp1
   exit
```

Filtering HTTPS Traffic

This section provides the commands for dropping or passing HTTPS traffic on any TCP port.

Dropping HTTPS Traffic on Any TCP Port

```
## ASF Configuration to pass HTTPS on Any Port

## Network Port Configuration example
## port 1/1/g1..g4 params admin enable
port <port-list> params admin enable

## Tool Port Configuration example
## port 1/1/g5 type tool
## port 1/1/g5 params admin enable
port <port-id> type tool
port <port-id> params admin enable

## GS group configuration example (GS group is created and associated to the GS engine)
## gsgroup alias gsgrp1 port-list 1/1/e1
gsgroup alias <alias> port-list <port-list>

## GS params configuration example (Buffer limits are set for the ASF application)
gsparams gsgroup gsgrp1 resource buffer-asf 2

## Reload GS Card example
# card slot 1/1 down
# no card slot 1/1 down
card slot <box ID>/<slot ID> down
no card slot <box ID>/<slot ID> down

## SAPF configuration example (Enabling / disabling buffering for sessions & session creating criteria)
apps asf alias https-sessions
   bi-directional enable
   buffer enable
   buffer-count-before-match 20
   sess-field add ipv4-5tuple outer
   exit

## Gsop configuration example (Gigasmart opertation for ASF function)
gsop alias https-gsop apf set asf https-sessions port-list gsgrp1
```

```
## Vport configuration example (Virtual port configuration to feed the egress traffic from first level maps)
vport alias vp1 gsgroup gsgrp1

## Traffic map connection configuration example (First level map to filter Web traffic)
##map alias ipv4traffic
##    rule add pass ipver 4 bidir
##    to vp1
##    from 1/1/g1..g4
##    exit
map alias ipv4traffic
   rule add pass ipver 4 bidir
   to vp1
   from <port-list>
   exit

## ASF lecond level map configuration to filter HTTPS traffic on any port example
##map alias https_filter
##    use gsop https-gsop
##    gsrule add pass pmatch protocol tcp pos 1 RegEx \x16\x03.{3}\x01 0..6
##    to 1/1/g5
##    from vp1
#    exit
map alias https_filter
   use gsop https-gsop
   gsrule add pass pmatch protocol tcp pos 1 RegEx \x16\x03.{3}\x01 0..6
   to <port-id>
   from vp1
   exit
```

Passing HTTPS Traffic on Any TCP Port

```
## ASF configuration to pass HTTPS on any port

## Network Port Configuration example
## port 1/1/g1..g4 params admin enable
port <port-list> params admin enable

## Tool Port Configuration example
## port 1/1/g5 type tool
## port 1/1/g5 params admin enable
port <port-id> type tool
port <port-id> params admin enable

## GS group configuration example (GS group is created and associated to the GS engine)
## gsgroup alias gsgrp1 port-list 1/1/e1
gsgroup alias gsgrp1 port-list <port-list>

## GS params configuration example (Buffer limits are set for the ASF application)
gsparams gsgroup gsgrp1 resource buffer-asf 2

## Reload GS Card example
# card slot 1/1 down
# no card slot 1/1 down
card slot <box ID>/<slot ID> down
no card slot <box ID>/<slot ID> down

## SAPF configuration example (Enabling / disabling buffering for sessions & session creating criteria)
apps asf alias https-sessions
   bi-directional enable
   buffer enable
   buffer-count-before-match 20
   sess-field add ipv4-5tuple outer
   exit

## Gsop configuration example (GigaSMART operation for ASF function)
gsop alias https-gsop apf set asf https-sessions port-list gsgrp1
```

```
## Vport configuration example (Virtual port configuration to feed the egress traffic from first level maps)
vport alias vp1 gsgroup gsgrp1

## Traffic map connection configuration example (First level map to filter Web traffic)
##map alias ipv4traffic
##    rule add pass ipver 4 bidir
##    to vp1
##    from 1/1/g1..g4
##    exit
map alias ipv4traffic
   rule add pass ipver 4 bidir
   to vp1
   from <port-list>
   exit

## ASF second level map configuration to filter HTTPS traffic on nny port example
##map alias https_filter
##    use gsop https-gsop
##    gsrule add pass pmatch protocol tcp pos 1 RegEx \x16\x03.{3}\x01 0..6
##    to 1/1/g5
##    from vp1
##    exit
map alias https_filter
   use gsop https-gsop
   gsrule add pass pmatch protocol tcp pos 1 RegEx \x16\x03.{3}\x01 0..6
   to <port-id>
   from vp1
   exit
```

Filtering Windows Update Traffic

This section provides the commands for dropping or passing the Windows update traffic.

Dropping Windows Update Traffic

```
## ASF configuration to drop Windows updates to tools / collector

## Network port configuration example
## port 1/1/g1..g4 params admin enable
port <port-list> params admin enable

## Tool port configuration example
## port 1/1/g5 type tool
## port 1/1/g5 params admin enable
port <port-id> type tool
port <port-id> params admin enable

## GS group configuration example (GS group is created and associated to the GS engine)
## gsgroup alias gsgrp1 port-list 1/1/e1
gsgroup alias gsgrp1 port-list <port-list>

## GS params configuration example (Buffer limits are set for the ASF application)
gsparams gsgroup gsgrp1 resource buffer-asf 2

## Reload GS card example
# card slot 1/1 down
# no card slot 1/1 down
card slot <box ID>/<slot ID> down
no card slot <boxId/slotId> down

## SAPF configuration example (Enabling / disabling buffering for sessions & session creating criteria)
apps asf alias winupdate-sessions
```

```
   bi-directional enable
   buffer enable
   buffer-count-before-match 20
   sess-field add ipv4-5tuple outer
   exit

## Gsop configuration example (GigaSMART operation for ASF function)
gsop alias winupdate-gsop apf set asf winupdate-sessions port-list gsgrp1

## Vport configuration example (Virtual port configuration to feed the egress traffic from first level maps)
vport alias vp1 gsgroup gsgrp1

## Traffic map connection configuration example (First level map to filter Web Traffic)
##map alias v4traffic
##    rule add pass ipver 4 bidir
##    to vp1
##    from 1/1/g6
##    exit
map alias v4traffic
   rule add pass ipver 4 bidir
   to vp1
   from <port-id>
   exit

## ASF second level map configuration to filter emails with attachments example
##map alias winupdates_filter
##    use gsop winupdate-gsop
##    gsrule add pass pmatch protocol tcp pos 1 RegEx  msdownload/update/software 0..1750
##    gsrule add pass pmatch protocol tcp pos 1 RegEx  download.windowsupdate.com 0..1750
##    to 1/1/g5
##    from vp1
##    exit
map alias winupdates_filter
   use gsop winupdate-gsop
   gsrule add pass pmatch protocol tcp pos 1 RegEx  msdownload/update/software 0..1750
   gsrule add pass pmatch protocol tcp pos 1 RegEx  download.windowsupdate.com 0..1750
   to <port-id>
   from vp1
   exit
```

Passing Windows Update Traffic

```
## ASF configuration to drop Windows updates to tools / collector

## Network port configuration example
## port 1/1/g6 params admin enable
port <port-list> params admin enable

## Tool Port Configuration example
## port 1/1/g5 type tool
## port 1/1/g5 params admin enable
port <port-id> type tool
port <port-id> params admin enable

## GS group configuration example (GS group is created and associated to the GS engine)
## gsgroup alias gsgrp1 port-list 1/1/e1
gsgroup alias gsgrp1 port-list <port-list>

## GS params configuration example (Buffer limits are set for the ASF application)
gsparams gsgroup gsgrp1 resource buffer-asf 2

## Reload GS Card example
# card slot 1/1 down
# no card slot 1/1 down
card slot <box ID>/<slot ID> down
no card slot <box ID>/<slot ID> down
```

```
## APF configuration example (Enabling / disabling buffering for sessions & session creating criteria)
apps asf alias winupdate-sessions
   bi-directional enable
   buffer enable
   buffer-count-before-match 20
   sess-field add ipv4-5tuple outer
   exit

## Gsop configuration example (GigaSMART operation for ASF function)
gsop alias winupdate-gsop apf set asf winupdate-sessions port-list gsgrp1

## Vport configuration example (Virtual port configuration to feed the egress traffic from first level maps)
vport alias vp1 gsgroup gsgrp1

## Traffic map connection configuration example (First level map to filter IPv4 traffic)
##map alias v4traffic
##    rule add pass ipver 4 bidir
##    to vp1
##    from 1/1/g6
##    exit
map alias v4traffic
   rule add pass ipver 4 bidir
   to vp1
   from <port-id>
   exit

## ASF second level map configuration to filter Windows updates example
##map alias winupdates_filter
##    use gsop winupdate-gsop
##    gsrule add pass pmatch protocol tcp pos 1 RegEx  msdownload/update/software 0..1750
##    gsrule add pass pmatch protocol tcp pos 1 RegEx  download.windowsupdate.com 0..1750
##    to 1/1/g5
##    from vp1
##    exit
map alias winupdates_filter
   use gsop winupdate-gsop
   gsrule add pass pmatch protocol tcp pos 1 RegEx  msdownload/update/software 0..1750
   gsrule add pass pmatch protocol tcp pos 1 RegEx  download.windowsupdate.com 0..1750
   to <port-id>
   from vp1
   exit
```

Filtering Email with Attachments

```
## ASF Configuration to drop emails with attachments to tools / collector

## Network Port Configuration example
## port 1/1/g6 params admin enable
port <port-list> params admin enable

## Tool Port Configuration example
## port 1/1/g5 type tool
## port 1/1/g5 params admin enable
port <port-id> type tool
port <port-id> params admin enable

## GS group configuration example (GS group is created and associated to the GS engine)
## gsgroup alias gsgrp1 port-list 1/1/e1
gsgroup alias gsgrp1 port-list <port-list>

## GS params configuration example (Buffer limits are set for the ASF application)
gsparams gsgroup gsgrp1 resource buffer-asf 2

## Reload GS Card example
# card slot 1/1 down
# no card slot 1/1 down
```

```
card slot <box ID/<slot ID> down
no card slot <box ID/<slot ID> down

## SAPF configuration example (Enabling / disabling buffering for sessions & session creating criteria)
apps asf alias email-sessions
   bi-directional enable
   buffer enable
   buffer-count-before-match 20
   sess-field add ipv4-5tuple outer
   exit

## Gsop configuration example (GigaSMART operation for ASF function)
gsop alias email-gsop apf set asf email-sessions port-list gsgrp1

## Vport configuration example (Virtual port configuration to feed the egress traffic from first level map)
vport alias vp1 gsgroup gsgrp1

## Traffic map connection configuration example (First level map to filter SMTP traffic)
##map alias smtptraffic
##    rule add pass portsrc 25 bidir
##    to vp1
##    from 1/1/g6
##    exit
map alias smtptraffic
   rule add pass portsrc 25 bidir
   to vp1
   from <port-id>
   exit

## ASF Second Level Map Configuration to Filter Emails with attachments example
##map alias email_filter
##    use gsop email-gsop
##    gsrule add drop pmatch protocol tcp pos 1 RegEx Content-Disposition 0..1750
##    to 1/1/g5
##    from vp1
##    exit
map alias email_filter
   use gsop email-gsop
   gsrule add drop pmatch protocol tcp pos 1 RegEx Content-Disposition 0..1750
   to <port-id>
   from vp1
   exit

## Pass rest of the traffic to the collector / tools example
##map-scollector alias collector
##    from vp1
##    collector 1/1/g5
##    exit
map-scollector alias collector
  from vp1
   collector <port-id>
   exit
```

Appendix B

Sample Gigamon-FM API Script

This appendix provides an example Python script that implements the recipes described in Chapter 3, *ASF with Gigamon-FM APIs*.

AsfTrafficFiltering.py

The section describes a Python script named `AsfTrafficFiltering.py` that can be used to implement the recipes with the GigaVUE-FM APIs. This script parses settings from a settings.txt file that are used in the requests. It also reads JSON files that contain map descriptions for the first and second level maps and a collector map. The settings file specifies which maps to use.

The script imports `apiclient.py`. which provides various functions for making and handling requests and responses. The `apiclient` import also includes some error handling.

`AsfTrafficFiltering.py` uses the libraries `ConfigParser`, `requests`, and `json`. If you do not already have them installed, you can install them with the `pip install` command. For example:

```
pip install requests
```

Figure 2-1 shows a flow chart of `AsfTrafficFiltering.py`. In cases where a a collector map is not use the script skips that step in the recipe.

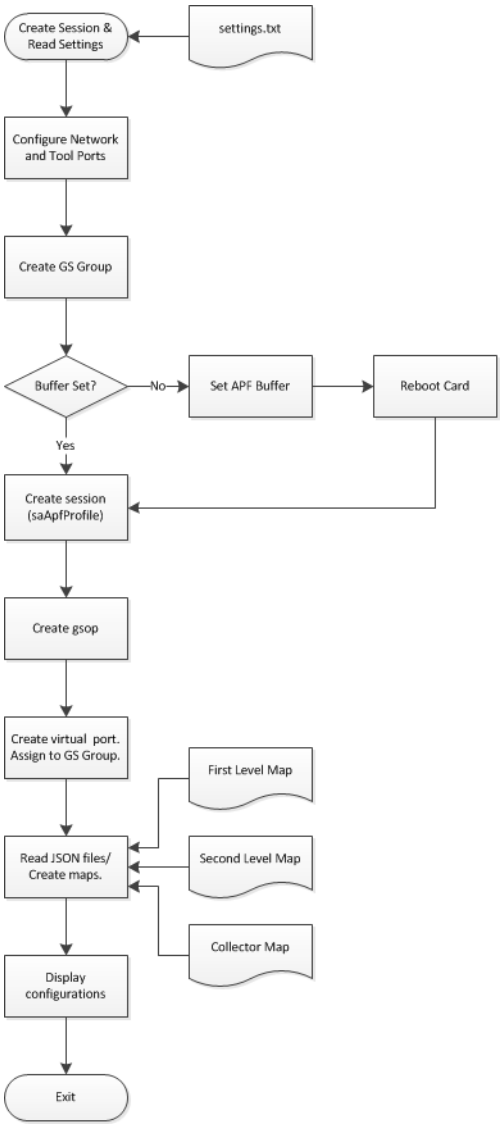

Figure 2-1: Flow Chart

```
#!/usr/bin/env python
#Copyright 2015 Gigamon Inc.

import ConfigParser
import requests
import json
import time

from requests.auth import HTTPBasicAuth
requests.packages.urllib3.disable_warnings()

import apiclient as api

def getMoreSettings(settingsFile):
    """ Returns the values from the settings file.

    Args:
        settingsFile: The path and filename of the settings file to parse.

    Returns:
        The additional values in the settings file as a tuple.
   """
```

```
    configParser = ConfigParser.RawConfigParser(allow_no_value=True)
    configParser.read(settingsFile)

    eport = configParser.get('settings', 'eport')
    vport = configParser.get('settings', 'vport')
    desc = configParser.get('settings', 'description')
    level1Mapfile = configParser.get('maps', 'firstLevelMap')
    level2Mapfile = configParser.get('maps', 'secondLevelMap')
    collectorMapFile = configParser.get('maps', 'collectorMap')

    return eport, vport, desc, level1Mapfile, level2Mapfile, collectorMapFile


def enablePort(url, session, portId):
    """ Enables the port by portId.

    Args:
        url: The URL for the PATCH request for enabling the port.
        session: The session object for the current request.
        portId: The ID of the port to enable.
    """

    payload = {
        "portId" : portId,
        "adminStatus" : "up"
    }

    print "Enabling port {}...\n".format(portId)
    print 'payload = {}\n'.format(json.dumps(payload, indent=4))
    print 'PATCH ' + url

    r = api.doPatch(url, payload, session, showHeaders=True)
    api.handleStatus(r)


def main():

    # Get the settings from the settings.txt file.
    (fmip, nodeip,
     networkport, toolport) = api.getSettings('c:\work\clicookbook\settings.txt')

    #Get custom settings from setting.txt file
    (eport, vPortAlias, desc,
     firstLevelMapFile,
     secondLevelMapfile,
     collectorMapFile) = getMoreSettings('c:\work\clicookbook\settings.txt')

    nporturl = networkport.replace('/','_')
    tporturl = toolport.replace('/','_')

    print 'FM IP Address - {}'.format(fmip)
    print 'Cluster IP Address - {}'.format(nodeip)
    print 'Using network port {}'.format(networkport)
    print 'Using tool port {}'.format(toolport)
    print 'Using engine port {}'.format(eport)
    print 'Using vport alias {}'.format(vPortAlias)
    print 'Using network port url {}'.format(nporturl)
    print 'Using tool port url {}'.format(tporturl)
    print
    print desc
    print
    print 'First level map {}'.format(firstLevelMapFile)
    print 'Second level map {}'.format(secondLevelMapfile)
    if collectorMapFile != None:
        print 'Collector map {}'.format(collectorMapFile)
    print

    #Create the session.
    session = requests.session()
    session.auth = ('admin', 'admin123A!')

    url = '{}/api/v1.1/nodes/flat?clusterId={}'.format(fmip, nodeip)
    print 'GET ' + url
```

```
r = api.doGet(url, session, showHeaders=True)
api.handleStatus(r, showContent=False)

if r.status_code == 200:
    nodeList = json.loads(r.text)
    nodes = nodeList['nodes']
    for node in nodes:
        if node['deviceIp'] == nodeip:
            print "  Hostname: {hostname}".format(**node)
            print "  Device ID: {deviceId}".format(**node)
            print "  Model: {model}".format(**node)
            print "  Cluster Mode: {clusterMode}".format(**node)
            print "  Software Version: {swVersion}".format(**node)

print
print "=============================="
step = 1
print 'Step {}. Configuring the network port.\n'.format(step)

print 'Checking whether port {} is a network port.'.format(networkport)

if api.isPortType(fmip, nodeip, session, networkport, 'network'):
    print 'Port {} is a network port.\n'.format(networkport)
    print 'Checking status'

    url = '%s/api/v1.1/inventory/ports/%s?clusterId=%s' % (fmip, nporturl, nodeip)
    r = api.doGet(url, session)
    api.handleStatus(r, showContent=False)

    portPayload = json.loads(r.text)
    port = portPayload['port']

    if port['adminStatus'] == 'down':
        print 'Network port {} is down'.format(networkport)
        enablePort(url, session, networkport)

    if port['adminStatus'] == 'up':
         print 'Network port {} is up'.format(networkport)

else:
    url = '{}/api/v1.1/inventory/ports/{}?clusterId={}'.format(fmip, nporturl, nodeip)

    payload = {
        "portId" : networkport,
        "alias" : "",
        "portType" : "network",
        "adminStatus" : "up"
    }

    print 'payload = {}\n'.format(json.dumps(payload, indent=4))
    print "Setting port  {} to a network port...\n".format(networkport)
    print 'PATCH ' + url

    r = api.doPatch(url, payload, session, showHeaders=True)
    api.handleStatus(r)

print 'Completed step {}\n'.format(step)

print "=============================="
step += 1
print 'Step {}. Configuring the tool port.\n'.format(step)

print 'Checking whether port {} is a tool port.'.format(toolport)

if api.isPortType(fmip, nodeip, session, toolport, 'tool'):
    print 'Port {} is a tool port.\n'.format(toolport)
    print 'Checking status'

    url = '%s/api/v1.1/inventory/ports/%s?clusterId=%s' % (fmip, tporturl, nodeip)
    r = api.doGet(url, session)
    api.handleStatus(r, showContent=False)

    portPayload = json.loads(r.text)
    port = portPayload['port']
```

```
        if port['adminStatus'] == 'down':
            print 'Tool port {} is down'.format(toolport)
            enablePort(url, session, toolport)

        if port['adminStatus'] == 'up':
             print 'Tool port {} is up'.format(networkport)

    else:
        url = '{}/api/v1.1/inventory/ports/{}?clusterId={}'.format(fmip, tporturl, nodeip)

        payload = {
            "portId" : toolport,
            "alias" : "",
            "portType" : "tool",
            "adminStatus" : "up"
        }

        print 'payload = {}\n'.format(json.dumps(payload, indent=4))
        print "Setting port  {} to a tool port...\n".format(toolport)
        print 'PATCH ' + url

        r = api.doPatch(url, payload, session, showHeaders=True)
        api.handleStatus(r)

    print 'Completed step {}\n'.format(step)

    print "=============================="
    step += 1
    print 'Step {}. Creating the GS Group.\n'.format(step)

    gsGroupAlias = "gsgrp-" + eport.replace('/','_')

    print "Checking whether GS Group already exists.\n"

    url = '{}/api/v1.1/gsGroups/{}?clusterId={}'.format(fmip, gsGroupAlias, nodeip)

    print 'GET ' + url
    r = api.doGet(url, session, showHeaders=True)
    api.handleStatus(r, exitOnFail=False, showContent=False)

    if r.status_code == 200:
        print 'GS Group %s exists.\n' % gsGroupAlias
    else:
        print 'GS Group {} does not exist.\n'.format(gsGroupAlias)
        url = '{}/api/v1.1/gsGroups?clusterId={}'.format(fmip, nodeip)

        payload = {
            "alias": gsGroupAlias,
            "ports": [eport]
        }

        print 'Configuring GigaSMART Group {} associated with port {}\n.'.format(gsGroupAlias,
eport)
        print 'payload = {}\n'.format(json.dumps(payload, indent=4))
        print 'POST ' + url

        r = api.doPost(url, payload, session, showHeaders=True)
        api.handleStatus(r)

    print 'Completed step {}\n'.format(step)

    print "=============================="
    step += 1
    print 'Step {}. Defining the maximum number of sessions.\n'.format(step)

    url = '{}/api/v1.1/gsGroups/{}?clusterId={}'.format(fmip, gsGroupAlias, nodeip)

    print 'Checking APF buffer.'
    print 'Get ' + url

    r = api.doGet(url, session, showHeaders=True)
    api.handleStatus(r, showContent=False)
```

```
gsGroupPayload = json.loads(r.text)
gsGroup = gsGroupPayload['gsGroup']
paramList = gsGroup['params']

doReboot = True
noBuffer = True
for param, value in paramList.items():
    if param == 'saApf':
        buffSize = value['bufferSize']
        if buffSize != 0:
            doReboot = False
            noBuffer = False

if not noBuffer:
    print 'APF buffer is set.'
    print 'bufferSize = {}'.format(buffSize)
else:
    print 'Creating APF buffer.'

    url = '{}/api/v1.1/gsGroups/{}/params?clusterId={}'.format(fmip, gsGroupAlias, nodeip)

    buffSize = 2
    payload = {
        "saApf":{
            "bufferSize": buffSize
        }
    }

    print 'Setting the number of sessions to {} million.\n'.format(buffSize)
    print 'payload = {}\n'.format(json.dumps(payload, indent=4))
    print 'PUT ' + url

    r = api.doPut(url, payload, session, showHeaders=True)
    api.handleStatus(r)

print 'Completed step {}\n'.format(step)

print "=============================="
step += 1
print 'Step {}. Reload the GigaSMART card.\n'.format(step)

if not doReboot:
    print 'APF buffer is set. Do not need to reboot.\n'
else:
    box = eport.split('/')[0]
    slot = eport.split('/')[1]
    slotId = box + '_' + slot

    url = '{}/api/v1.1/inventory/chassis/cards/{}?nodeId={}'.format(fmip, slotId, nodeip)

    adminStatus = ["down", "up"]
    for status in adminStatus:

        payload = {
            "slotId": slotId,
            "adminStatus": status
        }

        print 'payload = {}'.format(json.dumps(payload, indent=4))
        print 'PATCH ' + url
        print

        r = api.doPatch(url, payload, session)
        api.handleStatus(r)

print 'Completed step {}\n'.format(step)

print "=============================="
step += 1
print 'Step {}. Creating a flow session, specify the'.format(step)
print 'buffer count match, and enabling buffering.\n'
print "Checking if profile already exists.\n"

saApfProfilesAlias = "traffic-sessions"
```

```
    url = '{}/api/v1.1/apps/saApfProfiles/{}?clusterId={}'.format(fmip, saApfProfilesAlias,
nodeip)

    print 'GET ' + url
    r = api.doGet(url, session, showHeaders=True)
    print 'STATUS {} {}'.format(r.status_code, r.reason)

    if r.status_code == 200:
        print "Profile %s already exists.\n" % saApfProfilesAlias
    else:
        print "Profile %s does not exist.\n" % saApfProfilesAlias
        print 'Creating a flow session, specifing the buffer count before the match,'
        print 'and enabling buffering.\n'

        url = fmip + '/api/v1.1/apps/saApfProfiles?clusterId=' + nodeip

        payload = {
            "alias": saApfProfilesAlias,
            "sessionFields": [{
                    "pos": 1,
                    "type": "ipv4FiveTuple"
                }],
                "bidi": True,
                "buffering": {
                    "enabled": True,
                    "protocol": "tcpUdp",
                    "bufferCountBeforeMatch": 20
                }
            }

        print 'payload = {}\n'.format(json.dumps(payload, indent=4))
        print 'POST ' + url

        r = api.doPost(url, payload, session, showHeaders=True)
        api.handleStatus(r)

    print 'Completed step {}\n'.format(step)

    print "============================="
    step += 1
    print 'Step {}. Configuring a GigaSMART operation with APF.\n'.format(step)

    gsopAlias = "traffic-gsop"

    url = '{}/api/v1.1/gsops/{}?clusterId={}'.format(fmip, gsopAlias, nodeip)

    print 'Checking whether gsop already exists.\n'
    print 'GET ' + url

    r = api.doGet(url, session, showHeaders=True)
    print 'STATUS {} {}'.format(r.status_code, r.reason)

    if r.status_code == 200:
        print 'gsop {} exists'.format(gsopAlias)
    else:
        print 'gsop {} does not exist'.format(gsopAlias)
        url = '{}/api/v1.1/gsops?clusterId={}'.format(fmip, nodeip)

        payload = {
            "alias": gsopAlias,
            "gsGroup": gsGroupAlias,
            "gsApps": {
                "apf": {
                    "enabled": "enabled"
                },
                "saApf": {
                    "saApfProfile": saApfProfilesAlias
                }
            }
        }

        print 'Configuring a GigaSMART operation with APF.\n'
        print 'payload = {}\n'.format(json.dumps(payload, indent=4))
        print 'POST ' + url
```

```
    r = api.doPost(url, payload, session, showHeaders=True)
    api.handleStatus(r)

print 'Completed step {}\n'.format(step)

print "============================="
step += 1
print 'Step {}. Creating a virtual port and assign to GS Group.\n'.format(step)

print 'Checking whether virtual port exists.\n'

url = '{}/api/v1.1/vports/{}?clusterId={}'.format(fmip, vPortAlias, nodeip)
print 'GET ' + url

r = api.doGet(url, session, showHeaders=True)
print 'STATUS {} {}'.format(r.status_code, r.reason)

if r.status_code == 200:
    print 'Virtual port {} exists.\n'.format(vPortAlias)
else:
    print 'Virtual port {} does not exist.\n'.format(vPortAlias)

    url = '{}/api/v1.1/vports?clusterId={}'.format(fmip, nodeip)

    payload = {
        "alias": vPortAlias,
        "gsGroup": gsGroupAlias
    }

    print 'Creating virtual port ' + vPortAlias
    print 'payload = {}'.format(json.dumps(payload, indent=4))
    print 'POST ' + url

    r = api.doPost(url, payload, session, showHeaders=True)
    api.handleStatus(r)

print 'Completed step {}\n'.format(step)

print "============================="
step += 1
print 'Step {}. Creating a first level map to filter traffic.\n'.format(step)

#Loading map from file
with open(firstLevelMapFile) as json_file:
     payload = json.load(json_file)

#Get map alias from map. Update source and destination ports in map
firstLevelMapAlias = payload['alias']
payload["srcPorts"] = [networkport]
payload["dstPorts"] = [vPortAlias]

print 'Checking whether map {} exists.\n'.format(firstLevelMapAlias)

url ='{}/api/v1.1/maps/{}?clusterId={}'.format(fmip, firstLevelMapAlias, nodeip)
print 'GET ' + url

r = api.doGet(url, session, showHeaders=True)
print 'STATUS %d %s' % (r.status_code, r.reason)

if r.status_code == 200:
    print 'Map {} exists.\n'.format(firstLevelMapAlias)
else:
    print 'Map {} does not exist.\n'.format(firstLevelMapAlias)
    print 'Creating a first level map {}.\n'.format(firstLevelMapAlias)
    print 'payload = {}\n'.format(json.dumps(payload, indent=4))

    url = '{}/api/v1.1/maps?clusterId={}'.format(fmip, nodeip)
    print 'POST ' + url

    r = api.doPost(url, payload, session, showHeaders=True)
    api.handleStatus(r)

print 'Completed step {}\n'.format(step)
```

```
print "================================"
step += 1
print 'Step {}. Creating a second level map. The gsrule specifies'.format(step)
print 'the traffic to drop, using keywords. Buffered packets and'
print 'all subsequent packets will be dropped.\n'

url = '{}/api/v1.1/maps?clusterId={}'.format(fmip, nodeip)

#Loading map from file
with open(secondLevelMapfile) as json_file:
     payload = json.load(json_file)

#Get map alias from map. Update the source and destination ports in map.
secondLevelMapAlias = payload['alias']
payload["srcPorts"] = [vPortAlias]
payload["dstPorts"] = [toolport]

print 'Checking whether map {} exists.\n'.format(secondLevelMapAlias)

url = '{}/api/v1.1/maps/{}?clusterId={}'.format(fmip, secondLevelMapAlias, nodeip)
print 'GET ' + url

r = api.doGet(url, session, showHeaders=True)
print 'STATUS {} {}'.format(r.status_code, r.reason)

if r.status_code == 200:
    print 'Map {} exists.\n'.format(secondLevelMapAlias)
else:
    print 'Map {} does not exist.\n'.format(secondLevelMapAlias)
    print 'Creating a second level map {}.\n'.format(secondLevelMapAlias)
    print 'payload = {}\n'.format(json.dumps(payload, indent=4))

    url = '{}/api/v1.1/maps?clusterId={}'.format(fmip, nodeip)
    print 'POST ' + url

    r = api.doPost(url, payload, session, showHeaders=True)
    api.handleStatus(r)

print 'Completed step {}\n'.format(step)

#Skip the next step if a collector map is not used.
if collectorMapFile != None:
    print "================================"
    step += 1
    print 'Step {}. Creating a map for the collector.\n'.format(step)

    url = '{}/api/v1.1/maps?clusterId={}'.format(fmip, nodeip)

    #Loading map from file
    with open(collectorMapFile) as json_file:
        payload = json.load(json_file)

    #Get map alias from map. Update the source and destination ports in map.
    collectorMapAlias = payload['alias']
    payload["srcPorts"] = [vPortAlias]
    payload["dstPorts"] = [toolport]

    print 'Checking whether map {} already exists.\n'.format(collectorMapAlias)

    url = '{}/api/v1.1/maps/{}?clusterId={}'.format(fmip, collectorMapAlias, nodeip)
    print 'GET ' + url

    r = api.doGet(url, session, showHeaders=True)
    print 'STATUS {} {}'.format(r.status_code, r.reason)

    if r.status_code == 200:
        print 'Map {} exists.\n'.format(collectorMapAlias)
    else:
        print 'Map {} does not exist.\n'.format(collectorMapAlias)
        print 'Creating {} map.\n'.format(collectorMapAlias)
        print 'payload = {}\n'.format(json.dumps(payload, indent=4))

        url = '{}/api/v1.1/maps?clusterId={}'.format(fmip, nodeip)
```

```
            print 'POST ' + url

            r = api.doPost(url, payload, session, showHeaders=True)
            api.handleStatus(r)

        print 'Completed step %s\n' % step
    else:
        collectorMapAlias = None

    print "============================="
    step += 1
    print 'Step {}. Displaying configurations.\n'.format(step)

    #Set up url variables for use in urlList.
    #gsGroupUrl = fmip + '/api/v1.1/gsGroups/' + gsGroupAlias + '?clusterId=' + nodeip
    gsGroupUrl = '{}/api/v1.1/gsGroups/{}?clusterId={}'.format(fmip, gsGroupAlias, nodeip)
    saApfProfilesUrl = '{}/api/v1.1/apps/saApfProfiles/{}?clusterId={}'.format(fmip,
saApfProfilesAlias, nodeip)
    gsopsUrl = '{}/api/v1.1/gsops/{}?clusterId={}'.format(fmip, gsopAlias, nodeip)
    map1Url = '{}/api/v1.1/maps/{}?clusterId={}'.format(fmip, firstLevelMapAlias, nodeip)
    map2Url = '{}/api/v1.1/maps/{}?clusterId={}'.format(fmip, secondLevelMapAlias, nodeip)
    mapCollectorUrl =  '{}/api/v1.1/maps/{}?clusterId={}'.format(fmip, collectorMapAlias,
nodeip)

    urlList = [gsGroupUrl, saApfProfilesUrl, gsopsUrl, map1Url, mapCollectorUrl, map2Url]
    txtList = [gsGroupAlias, saApfProfilesAlias, gsopAlias, firstLevelMapAlias,
               collectorMapAlias, secondLevelMapAlias]

    for url, txt in zip(urlList, txtList):
        if txt != None:
            print 'Getting {} configuration\n'.format(txt)
            print 'GET {}'.format(url)
            r = api.doGet(url, session, showHeaders=True)
            api.handleStatus(r, exitOnFail=False)
            print

    print 'Completed step {}\n'.format(step)

    print "============================="

if __name__ == "__main__":
    main()
```

clientapi.py

The clientapi.py script is imported by `AsfTrafficFiltering.py`. It provides functions for making requests, a function for reading the settings.txt file, and a function for handling responses from requests.

```
#!/usr/bin/env python
#Copyright 2015 Gigamon Inc.

import json
import ConfigParser
import sys
import os
import webbrowser
import requests

pathname = os.path.dirname(sys.argv[0])
fullpath = os.path.abspath(pathname)

def handleStatus(resp, exitOnFail=True, showContent=True):
    """ Prints the status code from a response or the error message

    Args:
        resp: The Response from the request.
        exitOnFail: Optional variable that causes the application to exit when true.
            Otherwise the application continues after printing the error message.
```

```
        showContent: Optional variable that causes the function to print the
            content body of the response.
    """
    print 'STATUS %d %s' % (resp.status_code, resp.reason)
    if resp.status_code < 400:
        print 'SUCCESS\n'
        if showContent:
            print resp.content
            print
    else:
        print "Request failed..."
        try:
            gigaErrors = json.loads(resp.text)
            errors = gigaErrors['errors']
            for err in errors:
                print 'GigaError %s' % err['code']
                print err['msg'] + '\n'
            if exitOnFail:
                exit()
        except ValueError as e:
            print "Encountered an exception:"
            print '   %s\n' % e
            if resp.status_code > 200:
                print 'Exception details:'
                fname = fullpath + '\ErrorReport.html'
                f = open(fname, 'w')
                f.write(resp.text)
                f.close()
                print '   See %s\n' % fname
                webbrowser.open_new_tab(fname)
            if exitOnFail:
                print 'Exiting'
                exit()


def displayHeaders(header, payload=None):
    """ Prints out header information in a request.

    Args:
        header: The header from session.header.
        payload: The payload in the request. Use to calculate the
            length for the content-length header.
    """
    for headerType, headerValue in header.iteritems():
        if headerType == 'Content-Type' or headerType == 'Accept':
            print '%s : %s' % (headerType, headerValue)

    if payload is None:
        payload = []
    else:
        print 'Content-Length : %d\n' % len(json.dumps(payload))


def getSettings(settingsFile):
    """ Returns the values from the settings file.

    Args:
        settingsFile: The path and filename of the settings file to parse.

    Returns:
        The values in the settings file as a tuple.
   """
    configParser = ConfigParser.RawConfigParser(allow_no_value=True)
    configParser.read(settingsFile)

    fmip = configParser.get('settings', 'fmip')
    nodeip = configParser.get('settings', 'nodeip')
    nport = configParser.get('settings', 'nport')
    tport = configParser.get('settings', 'tport')

    return fmip, nodeip, nport, tport


def doPost(url, payload, session, showHeaders=False):
```

```
    """ Makes a post request and returns the response.

    Args:
        url: The url for the request.
        payload: The request body in json format.
        session: The session object for the current request.

    Returns:
        The response in json format.
    """
    session.headers.update({'Accept' : 'application/json'})
    session.headers.update({'Content-Type' : 'application/json'})
    resp = session.post(url, data=json.dumps(payload), verify=False)

    if showHeaders:
        displayHeaders(session.headers)

    return(resp)


def doPut(url, payload, session, showHeaders=False):
    """ Makes a put request and returns the response.

     Args:
        url: The url for the request.
        payload: The request body in json format.
        session: The session object for the current request.

    Returns:
        The response in json format.
    """
    session.headers.update({'Accept' : 'application/json'})
    session.headers.update({'Content-Type' : 'application/json'})
    resp = session.put(url, data=json.dumps(payload), verify=False)

    if showHeaders:
        displayHeaders(session.headers)

    return(resp)


def doPatch(url, payload, session, showHeaders=False):
    """ Makes a patch request and returns the response.

     Args:
        url: The url for the request.
        payload: The request body in json format.
        session: The session object for the current request.

    Returns:
        The response in json format.
    """
    session.headers.update({'Accept' : 'application/json'})
    session.headers.update({'Content-Type' : 'application/json'})
    resp = session.patch(url, data=json.dumps(payload), verify=False)

    if showHeaders:
        displayHeaders(session.headers)

    return(resp)


def doGet(url, session, showHeaders=False):
    """ Makes a get request and returns the response.

     Args:
        url: The url for the request.
        session: The session object for the current request.

    Returns:
        The response in json format.
    """
    session.headers.update({'Accept' : 'application/json'})
    resp = session.get(url, verify=False)
```

```
    if showHeaders:
        displayHeaders(session.headers)

    return(resp)


def doDelete(url, session, showHeaders=False):
    """ Makes a delete request and returns the response.
    Args:
        url: The url for the request.
        session: The session object for the current session.
    Returns:
        Returns the response in json format.
    """
    session.headers.update({'Accept' : 'application/json'})
    response = session.delete(url, verify=False)

    if showHeaders:
        displayHeaders(session.headers)

    return(response)


def isPortType(fmip, nodeip, session, nportId, portType):
    """ Checks whether a port is a network port by making a GET request.
    Args:
        fmip: The address of GigaVUE-FM.
        nodeip: The cluster ID used in the request.
        session: The session object for the current session.
        nportId: The ID of the port to check.
        portType: The type to check. The port types are 'network', 'tool', 'hybrid',
            'stack', 'inline-net', 'inline-tool', 'gateway', or 'gigasmart'.
    Returns:
        Returns True if port is of type specifed by portType. Otherwise, returns False.
    """

    portId = nportId.replace('/','_')
    url = '%s/api/v1.1/inventory/ports/%s?clusterId=%s' % (fmip, portId, nodeip)

    resp = doGet(url, session)
    handleStatus(resp, showContent=False)

    portPayload = json.loads(resp.text)
    port = portPayload['port']

    if port['portType'] == portType:
        return True
    else:
        return False
```

Settings File

The settings.txt file is a text file that contains information that the script uses, such as the GigaVUE-FM address (fmip), node and port ID, and the location of the files that contain the map payloads. This file is read by The following is an example of the contents of a settings file.

```
[settings]
fmip = https://TME-GFM-2
nodeip = 10.115.152.53
nport = 17/1/x1
tport = 17/1/x2
eport = 17/5/e1
vport = vp1
description = Filtering Netflix traffic
```

```
rebootwaitseconds = 90
[maps]
firstLevelMap = C:\work\clicookbook\WebTrafficMap.json
secondLevelMap = C:\work\clicookbook\NetflixDropMap.json
collectorMap = C:\work\clicookbook\CollectorMap.json
```

NOTE: In the example script, `ConfigParser` is set so that settings without values pass `None`. To skip the creation of a collector map, specify the `collectorMap` setting without a value.

Maps in JSON Format

This section provides the maps in JSON format that `AsfTrafficFiltering.py` reads from a .json file. In the maps, `scrPorts` and `dstPorts` do not specify any ports. Those values are read from the settings file and inserted by the script.

First Level Maps

This section provides the following first level maps for implementing the recipes:

- *Web Traffic Map* on page 87 is used in the recipes for filtering Netflix, YouTube, and Facebook traffic
- *IPv4 Traffic Map* on page 88 is used in the recipes for filtering HTTPS, email with attachements, and Windows traffic.
- *SMPT Traffic Map* on page 88 is used in the recipe for filtering VOIP traffic.

Web Traffic Map

```
{
   "alias" : "webtraffic",
   "type" : "firstLevel",
   "subType" : "byRule",
   "srcPorts" : [ ],
   "dstPorts" : [ ],
   "order" : 2,
   "rules" : {
      "passRules" : [ {
         "ruleId" : 1,
         "bidi" : true,
         "matches" : [ {
            "type" : "portDst",
            "value" : 80
         } ]
      }, {
         "ruleId" : 2,
         "bidi" : true,
         "matches" : [ {
            "type" : "portDst",
            "value" : 443
         } ]
      } ]
   }
}
```

IPv4 Traffic Map

```
{
   "alias" : "ipv4traffic",
   "type" : "firstLevel",
   "subType" : "byRule",
   "srcPorts" : [ ],
   "dstPorts" : [ ],
   "rules" : {
      "passRules" : [ {
         "ruleId" : 1,
         "bidi" : true,
         "matches" : [ {
            "type" : "ipVer",
            "value" : "v4"
         } ]
      } ]
   }
}
```

SMPT Traffic Map

```
{
   "alias" : "smtptraffic",
   "type" : "firstLevel",
   "subType" : "byRule",
   "srcPorts" : [ ],
   "dstPorts" : [ ],
   "rules" : {
      "passRules" : [ {
         "ruleId" : 1,
         "bidi" : true,
         "matches" : [ {
            "type" : "portSrc",
            "value" : 25

         } ]
      } ]
   }
}
```

Second Level Maps

This section provides the second level maps for the following recipes:

Recipe	Refer To
Filtering Netflix traffic	*Map for Dropping Netflix Traffic* on page 89
Filtering YouTube traffic	*Map for Dropping YouTube Traffic* on page 90
Filtering Facebook Traffic	*Map for Dropping Facebook Traffic* on page 91
Filtering HTTPS	*Map for Dropping HTTPS Traffic on Any Port* on page 91
Filtering Windows Update Traffic	*Map for Dropping Windows Update Traffic* on page 92
Filtering Email with Attachments	*Map for Passing Email with Attachments* on page 93

For the collector map used in the recipes, refer to *Collector Map* on page 93.

Map for Dropping Netflix Traffic

```
{
   "alias" : "netflix_filter",
      "clusterId" : "10.115.152.58",
      "type" : "secondLevel",
      "subType" : "byRule",
      "srcPorts" : [ ],
      "dstPorts" : [ ],
      "gsop" : "traffic-gsop",
      "gsRules" : {
         "dropRules" : [ {
            "ruleId" : 1,
            "matches" : [ {
               "type" : "pmatch",
               "value" :
"netflix|nflxvideo|nflximg|Netflix|nflxext|nflxsearch",
               "matchType" : "regex",
               "matchOffset" : {
                  "offsetStart" : 0,
                  "protocol" : {
                     "protocol" : "tcp",
                     "pos" : 1
               },
               "offsetEnd" : 1000
            }
         } ]
      }, {
         "ruleId" : 2,
         "matches" : [ {
            "type" : "pmatch",
            "value" : "octet-stream",
            "matchType" : "string",
            "matchOffset" : {
               "offsetStart" : 0,
               "protocol" : {
               "protocol" : "tcp",
               "pos" : 1
            },
            "offsetEnd" : 1000
         }
      } ]
   } ]
}
```

Map for Dropping YouTube Traffic

```
{
   "alias" : "youtube_filter",
   "clusterId" : "10.115.152.58",
   "type" : "secondLevel",
   "subType" : "byRule",
   "srcPorts" : [  ],
   "dstPorts" : [  ],
   "gsop" : "traffic-gsop",
   "gsRules" : {
      "dropRules" : [ {
         "ruleId" : 1,
         "matches" : [ {
            "type" : "pmatch",
```

```
                "value" : "youtube|ytimg|yt3.ggpht|tubeMogul|tmogulyoutu",
                "matchType" : "regex",
                "matchOffset" : {
                   "offsetStart" : 0,
                   "protocol" : {
                      "protocol" : "tcp",
                      "pos" : 1
                   },
                   "offsetEnd" : 1000
                }
             } ]
             }, {
                "ruleId" : 2,
                "matches" : [ {
                   "type" : "pmatch",
                   "value" : "Content-Type: video",
                   "matchType" : "string",
                   "matchOffset" : {
                      "offsetStart" : 0,
                      "protocol" : {
                         "protocol" : "tcp",
                         "pos" : 1
                       },
                      "offsetEnd" : 1000
                   }
                } ]
             }, {
                "ruleId" : 3,
                "matches" : [ {
                   "type" : "pmatch",
                   "value" : "octet-stream",
                   "matchType" : "string",
                   "matchOffset" : {
                      "offsetStart" : 0,
                      "protocol" : {
                         "protocol" : "tcp",
                         "pos" : 1
                      },
                      "offsetEnd" : 1750
                   }
                } ]
             } ]
          }
       }
```

Map for Dropping Facebook Traffic

```
       {
          "alias" : "fb_filter",
          "clusterId" : "10.115.152.58",
          "type" : "secondLevel",
          "subType" : "byRule",
          "srcPorts" : [ ],
          "dstPorts" : [ ],
          "gsop" : "traffic-gsop",
          "gsRules" : {
             "dropRules" : [ {
                "ruleId" : 1,
                "matches" : [ {
```

```
            "type" : "pmatch",
            "value" :
"^fdcdn.*\\.akamaihd\\.net|fbstatic.*\\.akamaihd\\.net|fbexternal.*\\.
akamaihd\\.net|.*\\.facebook\\.com|.*\\.fbcdn\\.net",
            "matchType" : "regex",
            "matchOffset" : {
            "offsetStart" : 0,
            "protocol" : {
               "protocol" : "tcp",
               "pos" : 1
            },
            "offsetEnd" : 1000
            }
         } ]
      } ]
   }
}
```

Map for Dropping HTTPS Traffic on Any Port

```
{
   "alias" : "https_filter",
   "type" : "secondLevel",
   "subType" : "byRule",
   "srcPorts" : [  ],
   "dstPorts" : [  ],
   "gsop" : "traffic-gsop",
   "gsRules" : {
      "passRules" : [ {
         "ruleId" : 1,
            "matches" : [ {
               "type" : "pmatch",
               "value" : "\\x16\\x03.{3}\\x01",
               "matchType" : "regex",
               "matchOffset" : {
                  "offsetStart" : 0,
                  "protocol" : {
                     "protocol" : "tcp",
                     "pos" : 1
                  },
                  "offsetEnd" : 6
               }
            } ]
      } ]
   }
}
```

Map for Dropping Windows Update Traffic

```
{
   "alias" : "winupdates_filter",
   "type" : "secondLevel",
   "subType" : "byRule",
   "srcPorts" : [  ],
   "dstPorts" : [ ],
```

```
   "gsop" : "traffic-gsop",
   "gsRules" : {
      "dropRules" : [ {
         "ruleId" : 1,
         "matches" : [ {
            "type" : "pmatch",
            "value" : "msdownload/update/software",
            "matchType" : "regex",
            "matchOffset" : {
               "offsetStart" : 0,
               "protocol" : {
                  "protocol" : "tcp",
                  "pos" : 1
               },
               "offsetEnd" : 1750
            }
         } ]
      }, {
         "ruleId" : 2,
         "matches" : [ {
            "type" : "pmatch",
            "value" : "download.windowsupdate.com",
            "matchType" : "regex",
            "matchOffset" : {
               "offsetStart" : 0,
               "protocol" : {
                  "protocol" : "tcp",
                  "pos" : 1
               },
               "offsetEnd" : 1750
            }
         } ]
      } ]
   }
}
```

Map for Passing Email with Attachments

```
{
   "alias" : "email_filter",
   "type" : "secondLevel",
   "subType" : "byRule",
   "srcPorts" : [ ],
   "dstPorts" : [ ],
   "gsop" : "traffic-gsop",
   "gsRules" : {
      "passRules" : [ {
         "ruleId" : 1,
         "matches" : [ {
            "type" : "pmatch",
            "value" : "Content-Disposition",
            "matchType" : "regex",
            "matchOffset" : {
               "offsetStart" : 0,
               "protocol" : {
```

```
                    "protocol" : "tcp",
                    "pos" : 1
                 },
                  "offsetEnd" : 1750
              }
           } ]
        } ]
     }
  }
```

Collector Map

```
  {
     "alias" : "collector",
     "type" : "secondLevel",
     "subType" : "collector",
     "srcPorts" : [ ],
     "dstPorts" : [ ]
  }
```

Appendix C

Additional Sources of Information

This appendix provides additional sources of information available from Gigamon.

Documentation

Gigamon provides additional documentation for the GigaVUE H Series on the Gigamon Customer Portal:

Document	Summary
GigaVUE HD Series Hardware Installation Guide	Describes how to unpack, assemble, rack-mount, connect, and perform the initial configuration of GigaVUE-HD8 and GigaVUE-HD4 nodes. Also provides reference information for these nodes, including specifications.
GigaVUE HC Series Hardware Installation Guide	Describes how to unpack, assemble, rack-mount, connect, and perform the initial configuration of GigaVUE-HC2 nodes. Also provides reference information for the GigaVUE-HC2 node, including specifications.
GigaVUE HB Series Hardware Installation Guide	Describes how to unpack, assemble, rack-mount, connect, and perform the initial configuration of GigaVUE-HB1 nodes. Also provides reference information for the GigaVUE-HB1 node, including specifications.
GigaVUE-OS CLI User's Guide	Describes how to configure and operate the GigaVUE-OS software from the command-line interface.
GigaVUE-OS H-VUE™ User's Guide	Describes how to use the web-based H-VUE interface to configure and operate the GigaVUE H Series software.
GigaVUE-OS H-VUE Online Help	Describes the web-based GUI for the H Series.
GigaVUE-OS Release Notes	Describes new features and known issues in the release.

Technical Support

Refer to http://www.gigamon.com/support-and-services/contact-support for Technical Support hours and contact information. You can also email Technical Support at support@gigamon.com.

Sales

The following is the contact information for the Sales Department at Gigamon.

Telephone +1 408.831.4025

inside.sales@gigamon.com

See Inside Your Network™