



# SSL Orchestrator

## Architecture Guide

*Version 14.0.0-4.0*



# Table of Contents

Table of Contents.....	3
Document Overview .....	6
General Architecture.....	7
Licensing and hardware Notes.....	7
Licensing and Provisioning.....	7
Compatibility / Support.....	8
Performance .....	13
Logging .....	13
Certificates .....	14
General Security Best Practices .....	15
Decoding Object Naming .....	15
Pre-Deployment Requirements .....	16
Basic Configuration Steps .....	17
Templated architecture concept .....	18
Deployment Architecture .....	23
Deployment scenarios .....	23
Deployment Mode .....	24
Deployment Topology Details.....	26
Service Architectures .....	34
Services Security Best Practices .....	34
Layer 2 Inline Service .....	35
Layer 3 Inline Device .....	37
Inline HTTP Proxy (Explicit / Transparent) .....	39
ICAP Device .....	41
TAP Device .....	43
Policy Architecture .....	45
Overall policy object relationships.....	45
SSL Settings Groups.....	45
SSLO Per-Request Policy .....	46
APM Per-Request Policies .....	47

Access Profile (Per-Session Policy).....	48
Interception Rules.....	48
Default APM Per-Request Policies .....	51
Base Policy .....	52
Categorization Macro .....	53
SSL Intercept Policy Macro.....	55
Service Chain Intercepted Macro.....	57
IP Policy Macro (Unused).....	57
Proxy Chaining (Connect / URI Rewrite) Macros (Unused) .....	59
Troubleshooting.....	62
Use Cases .....	67
Standard Use Cases.....	68
Block HTTP and HTTPS traffic based on URL.....	68
Select a different intercept chain based on IP intelligence or geolocation .....	70
Select the Non-Intercept chain based on IP information .....	74
HTTP should bypass all chains based on a user's group .....	78
Remap ports for inbound traffic .....	79
Categorize URLs for HTTP (non-encrypted) traffic.....	79
Setup without URL Filtering or SWG licensed.....	83
Configuring Inbound Services when SNAT is used.....	84
Configuring Layer 3 and HTTPS services for inbound traffic.....	85
Adding an iRule to a service.....	85
Adding a monitor to a TAP service.....	86
Configuring SSLO to use an upstream explicit proxy .....	87
Setting up authentication with explicit proxy connections .....	88
Setting policy to bypass ICAP services .....	90
Configure VLANs for L2 wire Interfaces .....	92
Advanced Use Cases .....	93
Create additional outbound explicit proxy rules .....	93
SNI for inbound listeners .....	94
Using SSLO in a horizontally scaled architecture .....	96
Provide IP address persistence for outbound traffic .....	98
Enable Client Certificate Constrained Delegation for SSLO .....	99

Add HTTP Headers to traffic .....	101
Configuring external communications for services .....	107
Appendixes.....	108
iRule: sslo_extras .....	108
Legal Notices .....	114

## Document Overview

SSL Orchestrator is a complex system that uses many different technologies and methods in combination to provide a secure solution for monitoring and orchestrating network traffic. The security level of the final deployed system is directly impacted by many variables, including the third-party monitoring systems, as well as the configuration of SSL Orchestrator and the network environment that it's deployed in.

This document is intended to provide an understanding of some of the architectures that are used by SSL Orchestrator v4.0. The following is an overview of what to expect in this document.

- This is intended to assist in system design and troubleshooting.
- There are places where the explanations may be summarized or simplified for clarity.
- This should be used in combination with any relevant BIG-IP documentation.
- This is not intended to provide step by step guidance on how to setup SSL Orchestrator.
- F5 Networks recommends that F5 Professional Services be consulted in any deployment.

# General Architecture

## Licensing and hardware Notes

- ! As of SSLO version 4.0, the following licensing options are no longer available:

  - Any Herculon platforms: (replaced with BIG-IP SSLO Standalone license).
  - VIPRION Licenses: version 4.0 does not support VIPRION.
  - SSLFWD (forward proxy) add-on license: Replaced with the SSLO Addon.

! Licenses generated prior to version 4.0 (BIG-IP v14.0.0) will not work with SSLO v4.0 and above.

! Customers can contact sales if they have any of the above (other than VIPRION) for licensing options to upgrade to 4.0.

SSLO version 4.0 is available via two licensing approaches;

- Standalone license: This license will license a platform that is dedicated to SSLO.
- Add-On license: This will add SSLO functionality to an existing LTM BIG-IP.

The standalone license is available for the following platforms:

- i2800
- i5800
- i11800
- i15800
- VE (8 CPU)
- VE (16 CPU)

The addon license can be used with the following platforms:

- 2xxx / i2xxx
- 4xxx / i4xxx
- 5xxx / i5xxx
- 7xxx / i7xxx
- 10xxx / i10xxx
- i11xxx
- i15xxx

The addon license can only be added to an LTM Base platform.

## Licensing and Provisioning

The following items must be provisioned for SSLO to operate correctly

<b>LTM</b>	Required for base functionality
<b>APM*</b>	
<b>SSLO</b>	

<b>AVR</b>	Required for Analytics to operate
<b>URLDB or SWG</b>	Required for URL filtering (you can deploy without this but will require extra actions to make it work).
<b>iRulesLX</b>	Optional, but will enable iRulesLX menu for troubleshooting.

- ! \*SSLO license includes a minimal limited for APM to allow SSLO features to run. The full APM license is not required, though it can be added-on to provide additional functionality.
- ! When SSLO is licensed, Access Guided Configuration is disabled, this does not impact any APM functionality, but users will not find the AGC menu option at this point.

The following additional licenses can be used for more filtering options:

<b>APM (Access Policy Manager)</b>	Provides the ability to create authentication profiles. supports SWG-Explicit profiles for forward proxy authentication, and normal access authentication methods for reverse proxy use cases. Transparent forward proxy (i.e. captive portal) authentication is not supported in any version yet.
<b>IPI (IP Intelligence)</b>	Provides access to the Webroot IP intelligence engine.
<b>NetHSM (network HSM)</b>	Provides access to hardware-based SSL private key storage for FIPS 140-2 L2 (or L3) level key protection.
<b>SWG (Secure Web Gateway)</b>	Provides granular URL and application-level access to the Websense/Forcepoint database (ex. Facebook chat vs. Facebook games), and other SWG functions, including malware detection. SWG includes URLF.
<b>URL (URL Filtering)</b>	Provides Host-level access to the Websense/Forcepoint database for making SSL bypass/intercept decisions.

- ! There is a known issue where the system will not process any traffic unless either URL filtering or SWG is licensed or provisioned. (See Setup without URL Filtering or SWG licensed for more information)

## Compatibility / Support

### Protocol Support

The following is a list of the protocols that SSLO supports:

Protocol	Detected (1)	Decrypted (2)	Inspected (3)
<b>HTTP/HTTPS</b>	Yes	Yes	Yes
<b>SMTP/SMTP over STARTTLS/SMTPS</b>	Yes	Yes	Yes
<b>POP3/POP3 over STARTTLS/POP3S</b>	Yes	Yes	Yes
<b>IMAP/IMAP over STARTTLS/IMAPS</b>	Yes	Yes	Yes
<b>FTP/FTPS (passive)</b>	Yes	Yes	Yes
<b>QUIC (TLS over UDP)</b>	Yes	No	No
<b>SSH</b>	Yes	No	Yes



<b>XMPP/XMPPS</b>	No (4)	No	Yes
<b>Other TCP non-TLS protocols (ex. LDAP, SIP, SNMP)</b>	No (4)	N/A	Yes
<b>Other TCP TLS-wrapped protocols (ex. LDAPS, SIP-TLS, SNMPS)</b>	No (4)	Yes	Yes
<b>Other UDP protocols</b>	No (4)	No	Yes
<b>Other non-TCP/non-UDP protocols</b>	No (4)	N/A	No

(1) Detected = SSLO can detect and make decisions based on this protocol

(2) Decrypted = SSLO can decrypt traffic using this protocol to send to the security services

(3) Inspected = SSLO can send traffic using this protocol to the security services

(4) Other protocols can alternatively be detected by "known" ports, via additional customization.

## What is forwarded to service devices

Traffic in SSLO can be handled in various ways depending on a variety of factors.

- HTTP verses Non-HTTP: Certain services cannot handle non-HTTP traffic. So, any Non-Http traffic will bypass these.
- If traffic is encrypted on receipt, and we can decrypt it, we can decrypt it before sending it to the service device. This happens by default but can be disabled if needed by using a non-intercepted chain. In some cases, traffic may be confidential enough that it should not be decrypted, but the admin does wish to send it to the services for accounting or later decryption.
- Traffic can also bypass all services completely if needed. TCP and TLS handshakes will always bypass service devices, but content traffic can as well if desired.

### Traffic Handling Table:

<b>Traffic Type</b>	<b>Initially Encrypted</b>	<b>Handling Mode</b>	<b>L2/L3 Inline</b>	<b>HTTP Inline</b>	<b>ICAP</b>	<b>TAP</b>
<b>TCP Handshake</b>	Any	Any	Bypass	Bypass	Bypass	Bypass
<b>TLS Handshake</b>	Any	Any	Bypass	Bypass	Bypass	Bypass
<b>HTTP</b>	Yes	Intercept	Clear	Clear	Clear	Clear
<b>Non-HTTP Traffic</b>	Yes	Intercept	Clear	Bypass	Bypass	Clear
<b>HTTP</b>	Yes	Non-Intercept	Encrypted	Encrypted	Encrypted	Encrypted
<b>Non-HTTP Traffic</b>	Yes	Non-Intercept	Encrypted	Bypass	Bypass	Encrypted
<b>HTTP</b>	No	Intercept	Clear	Clear	Clear	Clear
<b>Non-HTTP Traffic</b>	No	Intercept	Clear	Bypass	Bypass	Clear
<b>HTTP</b>	No	Non-Intercept	Clear	Clear	Clear	Clear
<b>Non-HTTP Traffic</b>	No	Non-Intercept	Clear	Bypass	Bypass	Clear

## TLS Version / Cipher Support

Most TLS ciphers and certificate types are supported, with the exception of the following:

Client Side	
<b>TLS 1.3</b>	TLS 1.3 should not be configured. The system will negotiate the best common TLS version available to both client and BIG-IP. If TLS 1.3 is configured and negotiated, it will not work.
<b>ECC and DSA signing</b>	SSLO does not support client side ECC and DSA signing. (ECC and DHE handshake algorithms are supported such as DHE-RSA and ECDHE-RSA), but DSA and ECDSA signing algorithms, which require a separate DSA or ECDSA signing key (ex. DHE-DSA, ECDHE-ECDSA) are not. <div><b>!</b> As a full proxy architecture this should not normally be an issue for customers, as server side ECC/DSA ciphers are supported.</div>
Server Side	
<b>TLS 1.3</b>	TLS 1.3 is not available for server SSL profiles at this point. The system will negotiate to TLS 1.2 connections as long as TLS 1.2 is configured and supported by the clients.

## Other Technology Support

### IMAP, SMTP and POP3 over STARTTLS

IMAP, SMTP and POP3 support STARTTLS, which is a technology to start TLS encryption from an already existing non-encrypted connection. This is normally handled by both sides advertising that they can handle TLS encryption, then one side will send a command to the other indicating that TLS encryption is desired, the sides will then negotiate the TLS connection and begin encrypting traffic. This can happen multiple times in a connection if desired.

SSL Orchestrator will detect and handle IMAP, SMTP, and POP3 connections which are unencrypted, as well as ones that are initially encrypted (IMAPS, SMTPS, and POP3S) in any normal transparent mode configuration. However, supporting STARTTLS for these requires that the application protocol handler is selected when the rule is created.

When STARTTLS is used, the unencrypted traffic is sent to the services, including any STARTTLS advertisements, however the command to initiate encryption and all of the SSL encryption handshaking will bypass the services. The encrypted data traffic will then be forwarded to the services after being decrypted.

If the protocol(s) are not selected in the rule(s), traffic will still be sent to the service devices with the following caveats.

- Unencrypted traffic will still be forwarded to the service devices.

- Fully encrypted connections will still be detected, decrypted and sent to the service device. (FTPS, IMAPS, SMTPS, etc.)
- STARTTLS connections will be detected as unencrypted and will be sent to the services devices without being decrypted, including encrypted traffic, however, the STARTTLS encryption will not operate correctly and the encryption portion of the connection will fail.

**!** Non-HTTP protocols will only work with transparent proxy implementations of SSL Orchestrator.

## FTP

FTP can work in active and passive modes, in both modes, the FTP client connects to the server on a command channel (normally port 20). In passive mode, the client asks the server for another port to use for data and the client then initiates another connection on that port. In active mode, the client tells the server what port to use, and the SERVER initiates a connection back to the client on the port specified.

**!** SSL Orchestrator can handle passive FTPS/FTP-STARTTLS connections only at this point. Active mode connections will fail.

In addition to the passive and active modes described above, FTP can operate in Implicit and Explicit modes. In Implicit mode, the client connects to the server and immediately initiates the TLS handshake and encryption. In explicit mode, the client connects to the server, and prior to exchanging data will initiate a conversion to TLS encrypted mode. This is also known as STARTTLS or Opportunistic TLS mode.

Normal FTPS and FTP-STARTTLS are both supported in passive mode.

STARTTLS is only supported when the FTP protocol handler is added to a rule. If the FTP protocol option is NOT selected in the rule(s), traffic will still be sent to the service devices with the following caveats.

- Unencrypted command and data traffic will still be forwarded to the service devices.
- Fully encrypted connections will also still be detected, decrypted and sent to the service device.
- FTP-STARTTLS connections will be detected as unencrypted and will be sent to the services devices without decryption, however, the STARTTLS encryption will not operate correctly and the encryption portion of the connection will fail.

**!** FTPS/FTP-STARTTLS will only work with SSLO rules operating in transparent mode.

## HSTS

HSTS will not impact SSLO operations.

### Certificate Pinning and HPKP

Certificate Pinning or HTTP Public Key Pinning is a technique where the server sends a set of the public keys directly to the browser in an HTTP header indicating which keys the browser should support. If the certificate that the browser receives does not match the key pinned in the HTTP header, the client must reject the connection. The exception to this for most browsers is if the certificate received is a known and trusted certificate, then the browser can trust it even if it does not match the pinned key.

<b>Browsers</b>	Certificate Pinning and HPKP should not impact browser-based connections as most browsers will ignore the pinning header if the certificate comes from a local trusted source.
<b>Non-Browser agents</b> (such as Dropbox apps, windows update, anti-virus update tools, etc.)	Non-browser-based agents may need to be configured to bypass SSLO as they often do not operate the same way. They often may not support adding an enterprise root CA to their trust store. In which case, they will use a built-in certificate, and fail to validate the SSLO one.

### Perfect Forward Secrecy (PFS)

Since SSLO terminates the SSL connections from both the client and the server, including Perfect Forward Secrecy connections, PFS use should not impact SSLO operations.

### QUIC (Most of Google's URL's)

QUIC is a UDP-based encrypted protocol (basically TLS over UDP). This is used by default by Chrome going to Google URLs. Service chaining and traffic rules do not work with QUIC, however the default UDP profile will return a reset upon receiving a QUIC connection, and most systems will then retry with normal TLS.

### Inline NAT/SNAT/PAT

Inline devices cannot NAT, SNAT, or PAT traffic. SSLO injects signaling for every packet that flows through, so that it can actively track traffic through dynamically-assigned services. Signaling for inline L3 and below devices uses ephemeral TCP (or UDP) information (source IP:port, destination IP:port) to track the packets. Any device, therefore, that changes source addresses (SNAT), destination addresses (NAT) or any of the ports (PAT) will destroy this signaling. A proxy device, for example, will always change the source port, and very often the source address, so must use an HTTP header for signaling. A proxy device, therefore, only works with HTTP traffic flows. Customers that have a NAT'ing firewall have a few options:

- Perform the NAT on the BIG-IP - outbound NAT rules are not usually that complex, so moving these to the F5 is trivial.
- Split the firewall into separate internal inspection and external NAT devices - many customers opt for this option as it's easiest to implement and keeps a firewall on the network edge.
- Architect a firewall "burrito" where traffic flows through the firewall twice - once decrypted (for inspection) and once encrypted (for NAT). This is a complex solution that requires advanced routing, so isn't generally recommended.

### Upstream systems that require a persistent source IP address.

SSLO does not apply any persistence to outbound route pools and SNAT pools, so the source IP address could change between connections. This could be a problem in situations such as one where a user goes to a site that performs federated authentication and tracks the source IP. Then when the client is redirected to the application, the IP changes and the user connection is blocked.

(See

! Provide IP address persistence for outbound traffic)

### Client Certificate Constrained Delegation (C3D)

C3D is not enabled by default with SSLO v4.0 virtual servers.

! (See Enable Client Certificate Constrained Delegation for SSLO)

## Performance

---

SSLO is a CPU-intensive process. Base performance can be about 40% of datasheet performance of the BIG-IP platform for TLS and bulk throughput, and 60% of datasheet performance for L4-L7 Application throughput.

Since all SSLO implementations will use one or more services, these services will also impact overall throughput, as well as any other systems they require.

For specific performance numbers, see the SSL Orchestrator Platform Datasheets.

## Logging

---

SSLO logs content into two main files;

<b>/var/log/ltn</b>	For normal logging, error logging, connections, and other normal BIG-IP logging.
<b>/var/log/apm</b>	For policy decisions. Most useful when troubleshooting problems with chains and services.

### Note

Debug logging will generate between up to several hundred messages for each request. So, it is critical that debug logging is only enabled for troubleshooting and immediately turned off.

### What is logged

It depends, but a lot. The primary difference between previous SSLO versions and 4.0 with regard to logging is the move from LTM to APM, so syslog information is now found in /var/log/apm. With debug enabled, you'll get at least:

- The client and server addresses and ports
- What protocol/cipher was negotiated on the client side
- What protocol/cipher was negotiated on the server side
- Every step through the security service policy (per-request policy)
- The ordered list of services that traffic flowed through, and whether or not it was decrypted
- This logging happens for every packet, so it is highly recommended not to run in debug mode in a production environment.

# Certificates

---

SSL Orchestrator forward proxy works by forging new certificates for clients based on the certificate received from the server. Reverse proxy works by sending its own certificate to the client. Because of this, understanding certificates is key to how SSL orchestrator works.

## General BIG-IP Certificates

---

<b>Management Certificate</b>	A self-signed certificate is created on setup to access the user interface. This can be replaced if desired with a normal certificate.
<b>Device Trust Certificate</b>	A self-signed certificate is created for config sync functions if a device group is used.

---

Note: Several other certificates are automatically created on the system and are used for secure communications between systems and for validating authorized F5 software.

## Forward Proxy Certificates

---

<b>Client certificate key chain</b>	This is the example certificate that is used as a base when forging certificates. Normally this would not need to be changed, but if custom flags or settings are needed, this can be updated to reflect that, and each forged certificate will mirror this. Normally this would simply use the default certificate and key.
<b>Client CA certificate</b>	This is the Certificate Authority certificate that is used to sign all forged certificates sent to the clients for transparent mode. This certificate must be distributed to all of the clients, otherwise they will get an “untrusted certificate” error. This certificate must be RSA based. This certificate must also have at least the following flags set: <ul style="list-style-type: none"><li>• Digital Signature key usage (digitalSignature)</li><li>• Certificate Signing key usage (keyCertSign)</li><li>• CA Basic Constraint set to TRUE</li></ul>
<b>Server Trusted CA Bundle</b>	This is the certificate bundle that is used to validate your servers. It must include the root CA for all servers that clients will be accessing. Often this is a custom bundle that includes the corporate CA.

---

## Reverse Proxy Certificates

---

<b>Client Certificate key chain</b>	This is the certificate used to send to the clients on connection. It should be one that the clients will trust, so if it is a custom certificate, then the root must be distributed to the clients, or it can be a commercial certificate based on one of the trusted certificate providers.
-------------------------------------	---

---

<b>Server Trusted CA Bundle</b>	<p>This is the certificate bundle that is used to validate your servers. It must include the root CA for all servers that clients will be accessing.</p> <p>This often uses the default “ca-bundle.crt” bundle which is maintained by F5 to include the normal root certificates.</p>
---------------------------------	---

## General Security Best Practices

### Avoid Bypass on Failure

The Bypass on Handshake Failure and Bypass on Client Cert Failure should be used with caution. It is possible that a colluding client and server could bypass inspection by forcing TLS errors.

### Avoid AutoMap

Never use auto-map for the gateway route.

### Avoid Debug Logging

Never leave Debug logging enabled.

## Decoding Object Naming

A normal SSL Orchestrator deployment can generate over a hundred TMOS objects. These are named using a scheme to help determine what each of them is used for.

**!** Not all objects follow the below naming rules, so care should be used in modifying any objects in TMOS unless it's clear that it is not an SSLO object.

### General Guidelines

- Most objects are named starting with “sslo”
- Immediately after the “sslo” may be a character indicating what SSLO object this is related to:
  - o ssloS: Service Objects.
  - o ssloP: Policy Objects.
  - o ssloN: Network Objects.
  - o ssloT: SSL Group objects.
  - o sslo: (no extra character) indicate base objects, as well as interception rules and related objects.
- Objects that are specific to TCP or UDP will have a “-t-” or “-u-” in the name.
- Objects that are specific to IPv4 or IPv6 will have a “-4-” or “-6-” in the name.

### Virtual Server Guidelines

- There are two types of virtual server objects;
  - o Service objects
    - Start with “ssloS\_<service name>”
    - Outbound objects (send traffic to the service) only have the t/u and/or 4/6 after the service name.
    - Inbound objects (receive traffic from the service) have a “-D-” in the name

- ICAP objects can have “-req” and “-res” after the name to designate servers to handle requests and responses.
- Tap objects do not have an inbound object.
- Proxy objects
  - start with “sslo\_<sslo application name>”
  - after the application name will be a code indicating what type of traffic it handles:
    - Application specific traffic: “-ftp-”, “-ftps-”, “-imap-”, “-pop3-”, “-smtp25-”, “-smtp257-”.
    - Protocol specific traffic:
      - “-in-”: handles TCP or UDP traffic
      - “-ot-”: handles non-TCP/UDP traffic
      - “-xp-”: handles http explicit proxy traffic.
- Profiles and other associated objects will normally start with “ssloS\_<service name>” or “sslo\_<sslo application name>” and sometimes a string indicating its purpose.

## Pre-Deployment Requirements

---

Prior to deploying SSL Orchestrator, you should have the following tasks and objects configured in BIG-IP.

Some of these can be configured during the device setup wizard and some need to be completed after the wizard is finished.

### General Items

Internal and External VLANS and Self IP's	*Required
NTP and time zone settings	*Required
DNS	*Required
All required VLANS (for vCMP implementations)	*Required for vCMP client Implementations
DNSEC	If desired
Device Groups and HA	Config sync should be set to Manual if enabled.
Certificates created and installed.	(See Certificates section.) Also any certificate management items such as CRL and OCSP.
Cipher Groups	If desired
Custom URL and IPI groups	If desired
ICAP VLAN and Self IP	If ICAP is needed.



## Basic Configuration Steps

---

While SSLO CAN be installed into a pre-existing network design, it is recommended that you should redesign how your security devices are connected to the network, as well as the addressing of all security devices that will be used.

In the Service Architectures section below, each service has a note about implementation design and required information. Pay attention to this when designing the deployment as some services need to be addressed before starting, and others will use addresses that must be in a subnet that will be provided by SSLO on configuration.

The high-level recommended approach to installing and configuring SSLO is as follows:

**!** Note that this approach does NOT follow the order of the items in menus within SSLO

1. Design the system before starting.
  - a. Determine if you will be deploying SSLO using a Layer 2 or Layer 3 deployment mode (See: Deployment Mode)
  - b. Determine the connectivity type(s) and direction of traffic needed. (See: [Deployment Topology Details](#))
  - c. Determine the list of services that you will be using.
  - d. Determine how each service will be connected to the network.(See: Service Architectures) and collect the required information.
2. Install SSL Orchestrator in the rack and connect to the network and services. (See platform guide)
3. After power on, complete the initial setup wizard.
4. Collect all certificates that will be needed. (See: Certificates)
5. Configure any required BIG-IP items before starting the SSL Orchestrator configuration. (See: [Pre-Deployment Requirements](#))
6. Configure the deployment settings [SSL Orchestrator / Deployment / Deployment Settings] (See SSL Orchestrator Reference Guide)
7. Configure specific services that will be used [SSL Orchestrator / Deployment / Services / ...] (See: SSL Orchestrator Reference Guide)
  - a. Configure any remaining service device addressing from the defined services objects.
8. Create the default set of interception rules. [SSL Orchestrator / Deployment / Interception Rules] (See: Interception Rules)

Add and customize rules and per-request policies as needed. (See:

## 9. SSLO Per-Request Policy)

For information on configuring specific SSL Orchestrator objects, see the SSL Orchestrator v4.0 Reference Guide.

# Templated architecture concept


SSL Orchestrator uses a different approach to configuration that combines multiple iAppLX templates into one interface.

It is still deployed using one iAppLX Template however, upon deployment, it uses multiple sub templates for individual sections. This allows for smaller chunks of configurations to be “protected” by the strict update field. It also allows for faster configuration changes.

### Sub templates:

<b>Deployment</b>	Main configuration
<b>Policy</b>	Policy, rules, and chains.
<b>Services</b>	Service configurations
<b>Network (groups)</b>	VLAN and Address objects
<b>SSL (Setting groups)</b>	SSL Setting Groups

As each template is used in SSL Orchestrator, it will generate a BIG-IP configuration with a set of objects to meet the templates selections. On most configuration pages in SSL Orchestrator there is a “strict update” option which will lock these created objects and disallow anyone from modifying them outside of SSL Orchestrator. If someone tried to do so, they will receive an error like this:

 010715bc:3: The application service (/Common/ssloS\_http\_exp.app/ssloS\_http\_exp) has strict updates enabled, the object (Virtual Server /Common/ssloS\_http\_exp.app/ssloS\_http\_exp-t) must be updated using an application management interface.

This indicates that the object you are trying to edit is locked and must be managed from the SSL Orchestrator object.

! The SSL Orchestrator v4.0 Reference Guide documents which types of BIG-IP objects are managed by each SSL Orchestrator template type.

Due to the approach that SSL Orchestrator takes in managing configuration objects, there are several different approaches that may be needed for any given configuration need. Some of these have a higher level of potential impact.

Examples of different common customizations and use cases can be found in the Use Cases section.

Policy objects do not have a “strict update” option as they are intended to be modified by administrators.

! As always, you should also make sure to back up your system before starting any modifications.

## Simple modifications and configurations

Simple customizations include making changes that are fully supported and documented, and any changes to objects that are deployed by SSL Orchestrator are done from within the SSL Orchestrator interface.

### These include:

- Using the SSL Orchestrator Interface.
- Modifying BIG-IP objects or configurations not deployed by SSL Orchestrator.

### Potential Issues:

These modifications and customizations do not add any additional issues beyond normal configuration changes.

## Complex modifications and configurations

Complex modifications may include making changes to visual policies that were deployed by SSL Orchestrator, as well as other modifications that directly impact objects that are created by SSL Orchestrator.

### Changing Interception rules and Interception rule objects

The objects that are created by creating and modifying rules, especially the APM Per-Request policies are a little more stable and are not protected by the “Strict update” option. However, they can also be over-written when modifying the default rules or the SSLO Per-Request policy (chain) object, and therefore should also be checked after making any modifications.

In most implementations there will only be two APM Per-Request policies, one for TCP and one for UDP. And in many cases you will only have to modify the TCP policy when the modification only concerns TCP (such as when it is for HTTP or HTTPS traffic).

In addition, it is possible to create additional SSLO Per-Request policies, which result in additional APM Per-Request policies. In these cases, any modification may have to be done on these additional policies as well if they are in use by rules.

### These include:

- Modifying the Per-Request Policy created by SSL Orchestrator in VPE
- Creating custom per-request visual policies for SSL Orchestrator
- Adding new iRules to objects managed by SSL Orchestrator

### Potential Issues:

- ! Per-Request Policies that are created when default rules are created may be modified, however, editing the SSLO configuration may overwrite the changes to the policy.
- ! Creating new per request policies is supported, however, after creation, modification of services and options may not be reflected in custom per-request policies.

! Adding custom iRules to some SSL Orchestrator objects is supported (though attaching rules to service objects require some additional steps, see Adding an iRule to a service). F5 technical support however will only provide best effort support for custom iRules. (see F5 support policy for more information). The exception to this is official workarounds that are implemented as documented. If a documented workaround includes adding a custom iRule, that iRule will be considered supported.

## Modification of SSLO deployed objects

BIG-IP is a very flexible platform for traffic management, and there are many ways that BIG-IP objects can be configured to address different needs. However, SSL Orchestrator configures these objects in very specific ways to provide security, reliability, and performance in handling sensitive information. Modifying BIG-IP objects that were created by SSL Orchestrator includes making any changes to objects that were deployed by SSL Orchestrator. These objects are protected by the “Strict Update” option which must be disabled before modifying them. Modifying anything protected by “Strict Update” is not recommended and may have caused unknown issues.

This includes recommendations in other BIG-IP manuals or modules that are not specifically related to SSL Orchestrator.

***After disabling ‘Strict Update’, if you need to make further changes to the SSLO configuration for that object from within SSL Orchestrator, you will need to take the following actions:***

1. ***Document your prior modifications.***
2. ***Re-enable “Strict Update” (THIS WILL OVERWRITE YOUR PRIOR MODIFICATIONS!)***
3. ***Update SSL Orchestrator as needed.***
4. ***Disable “Strict Update”***
5. ***Re-Implement the prior modifications.***

### These include:

- Modifying Virtual Servers, Policies, Profiles, and other objects deployed by SSLO (even if documented by F5, if not specifically for SSL Orchestrator)
- Modification of any iRules that were deployed by SSL Orchestrator.

### Potential Issues:

- ! Modification of any deployed objects may cause significant un-intended consequences and security issues.
- ! Any modifications to SSLO deployed BIG-IP objects WILL be overwritten when strict update is later re-enabled. In addition, if strict update is not re-enabled and SSLO objects were updated, the resulting deployment might cause un-expected behaviors, including requiring a complete system reinstall.
- ! Modifications of any deployed objects not considered a supported configuration action. F5 technical support will only provide best effort support for solutions that have modified their system. (see F5 support policy for more information). The exception to this is official workarounds that are implemented as documented. If a documented workaround includes modifying deployed objects, those changes are considered supported.

## Prohibited Modifications

Prohibited modifications and customization include making any changes to the underlying BIG-IP scripts, SSL Orchestrator iAppLX scripts or package, or any other Linux scripts within BIG-IP.

### These include:

- Making changes to BIG-IP scripts.
- Making changes to the SSL Orchestrator iAppLX scripts or package.
- Modifying any SSL Orchestrator deployed iRules
- Making any other unauthorized changes to the underlying operating system of the BIG-IP.

### Potential Issues:

! Modifications of any underlying scripts or operating system files is considered a prohibited modification. F5 technical support will not provide support for systems that have been so modified. (see F5 support policy for more information). The exception to this is official workarounds that are implemented as documented. If a documented workaround includes modifying these items, those changes are considered supported.

## Where to add iRules

One of the things that makes BIG-IP so popular is the ability to interact with traffic using iRules. SSL Orchestrator has several places to add iRules if needed.

<b>In Services</b>	These would run each time the service is used in a chain. Note there is a known issue where you cannot add an iRule to a service, see Adding an iRule to a service )
<b>In Interception Rules</b>	These would run each time a chain is run, prior to the traffic starting the chain
<b>Attached to virtual Servers</b>	These would run based on the server they are attached to, see the Categorize URLs for HTTP (non-encrypted) traffic use case for an example of this.
<b>In the APM Per-Request Policy</b>	These would run once per request, depending on where in the policy they were attached.

## Where to modify a policy

Before adding branches and actions, you need to determine where the information you need to make the decision will be available.

Using the default TCP policy for example, decryption (if needed) does not happen until after the SSL Intercept policy, so you would be unable to make a decision based on anything that is encrypted such as HTML or HTTP information.

The below diagram shows some of the common information types available at different points of the policy. This information may change as it moves through each agent in each of the individual macros, so careful attention to what each is doing is critical. See

Default APM Per-Request Policies for more information on each step and its agents.



1. At this point we have just received the traffic, so we know the IP address and TCP / UDP port information, and if the traffic is not encrypted, we will probably be able to look deeper at the content for things like URI and HTTP headers.
2. After the Categorization step, we have the URL category available, but still do not have anything that is unencrypted available such as HTTP headers.
3. After the SSL Intercept Policy (actually after the “SSL Intercept Set” agent) in the SSL Intercept Policy, we will have the decrypted content available for decisions.
4. After leaving the Service Chain, the content may have been modified by the individual agents, at this point you can see the final content before it is re-encrypted and sent to the destination.

Taking this into account is critical in designing changes to the policy.

As an example, if we tried to determine a branch, or block based on the URL category of some HTTPS traffic, we would not be able to do that until after it was categorized, which happens in the Categorization macro. So, we would need to add this step in the SSL Intercept Policy Macro (and you can see an example of this already in in that macro).

Also, keep in mind that you may need to be able to handle failures as well. Look at the default IP Policy macro for an example of this with the IP Reputation lookup. In that case, if the IP reputation lookup failed, the system will continue to process the traffic. This may or may not be the case for your implementation, but not taking it into account may result in unexpected behavior.

# Deployment Architecture

## Deployment scenarios

SSLO is deployed as an inline device on the network and supports both single devices and HA clusters using device groups.

! Note that SSLO 4.0 **DOES NOT** support the two-box configuration model (separate decrypt and re-encrypt boxes) like previous versions. SSLO 3.0 and the FC version still support a two-box configuration, and support for this is on the SSLO roadmap (7.0+).

A deployment topology is the combination of the way SSL Orchestrator sits on the network and how it processes incoming traffic;

Deployment topology can be described by a combination of;

- SSL Orchestrator's deployment mode (layer 2 or layer 3)
- Direction of traffic flow (inbound or outbound)
- SSLO's proxy type (transparent, explicit, or both)

Any given SSL Orchestrator can only be deployed in either a layer 2 mode, or a layer 3 mode, but it can use multiple combinations of traffic flow and proxy type.

SSLO L3 Services and HTTP services will not work if multiple SSLO implementations are load balanced behind another load balancer. (See

! Using SSLO in a horizontally scaled architecture)

### External Proxy Device

In addition to the ability to send traffic through a HTTP proxy as a service device (discussed later), SSL Orchestrator can support having proxies before and after the system.

SSLO supports both transparent and explicit proxy devices both before and after the system, however explicit proxies that are AFTER SSLO will need additional configuration of SSLO to correctly send traffic to them.

In addition, SSLO does not support authentication with explicit proxy devices setup after system.

1. To setup SSLO for use with an upstream explicit proxy, you must modify the policy after it is generated. (See

! Configuring SSLO to use an upstream explicit proxy)

! SSLO does not support an upstream Explicit proxy device when SSLO itself is configured as a transparent proxy device.

# Deployment Mode

The deployment mode is used to configure how the SSLO interacts with the network around it. This can be set to either Layer 2 Network (if the hardware supports it) or Layer 3 Network.

## Layer 2 Network:

This operates as a transparent switch in the network and must be installed between two VLANs in such a way that the traffic must flow through the system.

This approach, called a "virtual wire" topology mode (SSLO as a bump-in-the-wire) is only available on (i5800, i7800, i10800, i11800, i15800 as of SSLO v4.0). This mode is enabled with special characteristics of the Broadcom network chipsets which enable full L2 header passing from ingress to egress.

There is still a full proxy environment inside the inspection zone, allowing L3, ICAP and proxy devices to function, so this new mode isn't technically a "true" bump-in-the-wire, but does not require any L3 addressing on the edges.

- ! Virtual wire can only be created
- ! on select platforms with a Broadcom chip. SSLO application will correspondingly only allow L2 wire deployment on these platforms.
- ! Only transparent proxy mode is supported by SSLO for Layer 2 Network deployments.
- ! Care should be taken in connecting to the network, since, until SSLO is configured, the system will forward all traffic, including broadcast traffic, which could cause a broadcast storm.
- ! The Layer 2 header will not be sent to services by SSLO.
- ! The default gateway and SNAT are no longer configurable.
- ! Allow non-UDP/non-TCP in rules are selected by default cannot be unchecked.
- ! Available VLANs are limited to the VLAN groups having Transparency mode as Virtual Wire.

## Setting up SSLO for a Layer 2 Network

Before selecting a Layer2 Network, there are some extra steps that must be taken to configure the BIG-IP.

SSLO will automatically set some internal database variables after Layer 2 is selected. If VWire is not working these should be checked however using "tmsh list sys db <name>":

<b>vlangroup.forwarding.override</b>	Overrides VLAN group forwarding to allow forwarding of packets through the wildcard virtual (Should be "disable")
<b>bcm5x66x.vwire.4kvlan</b>	Allows dynamic creation of VLANs under the virtual wire (Should be "enable")

### Configuration Steps:

1. Modify port-forwarding mode for interfaces that are part of the L2 wire

The command for this is:

```
modify net interface <interface> port-fwd-mode virtual-wire
```



Example:

```
modify net interface 1.1 port-fwd-mode virtual-wire
modify net interface 1.2 port-fwd-mode virtual wire
```

2. Configure VLANs on the L2 wire interfaces with the tag 4096 (for "any" VLAN)

The command for this is:

```
Create net vlan <vlan name> tag 4096 interfaces add { <interface> { tagged } }
```

Example:

```
create net vlan l2wire-1-any tag 4096 interfaces add { 1.1 { tagged } }
create net vlan l2wire-2-any tag 4096 interfaces add { 1.2 { tagged } }
```

3. Configure a VLAN-group linking the L2 wire interfaces

The command for this is:

```
create net vlan-group <group_name> members add { <vlan name> ... } mode virtual-wire
```

Example:

```
create net vlan-group l2wire-any-vg members add { l2wire-1-any l2wire-2-any } mode virtual-wire
```

4. Configure VLANs for L2 wire Interfaces

You can create VLANs using either static VLANs or Dynamic VLANs in this configuration. F5 recommends creating static VLANs for the expected traffic.

- a. Static VLANs

Static VLANs can be configured under the L2 wire interfaces *after* the base configuration of the "any" VLAN by creating VLANs for the specific tags and then creating a VLAN-group for those VLANs.

The command for this is:

```
create net vlan tag <vlan id> interfaces add { <interface> { tagged } }
create net vlan-group <vlan group> members add { <vlan name> ... } mode virtual-wire
```

Example:

```
create net vlan l2wire-1-100 tag 100 interfaces add { 1.1 { tagged } }
create net vlan l2wire-2-100 tag 100 interfaces add { 1.2 { tagged } }
create net vlan-group l2wire-100 members add { l2wire-1-100 l2wire-2-100 } mode virtual-wire
```

- b. Dynamic VLAN creation under L2 Wire

Any VLAN traffic received on a that has a VLAN that is not statically created and seen under a L2wire port are dynamically learned, and internal VLANs and VLAN-group are automatically created for the dynamic VLANs (similar to static configuration).

! Only two VLANs can be part of a the L2 wire, however, the VLAN itself could be built on top of a trunk-interface which has multiple physical interfaces.

! You can have more than one L2 wire on the BIG-IP system

## Layer 3 Network:

This deployment mode operates as a router in the network. If using with transparent mode, downstream routers consider the SSLO as a default gateway to the internet, and upstream routers routing through it to get to the internal resources.

If operating as an explicit proxy, upstream and downstream routers must simply have a route to get to the SSLO system.

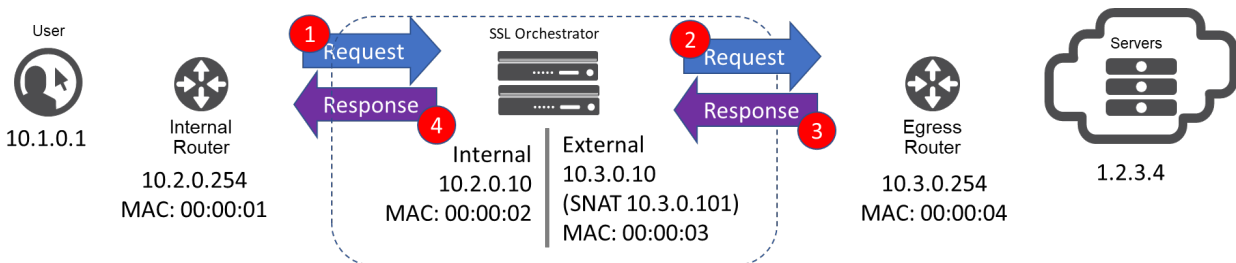
In most cases this is the best option to choose as it provides the most flexibility.

## Deployment Topology Details

In addition to the deployment mode, a system can be configured to handle inbound and outbound traffic, and in the case of one configured for Layer 3 mode, it can handle both explicit and transparent proxies. L2 mode can only support transparent proxy deployment.

### Layer 3, Outbound transparent proxy

This topology provides a transparent outbound, or forward proxy solution to monitor traffic from internal users and systems going to external systems. With this topology, clients do not need to be configured with the SSLO as a proxy for their systems.



### Addressing Example:

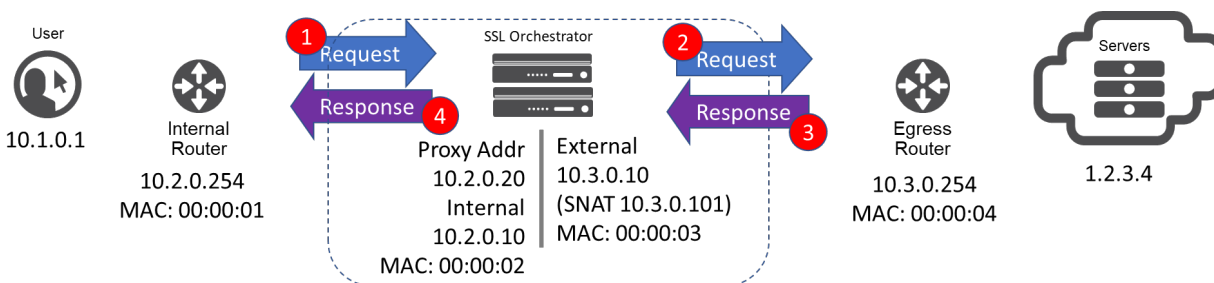
Step	Source IP	Dest IP	Dest MAC
1	10.1.0.1 (Client)	1.2.3.4 (Server)	00:00:02 (SSLO Int)
2	10.3.0.101 (SNAT)	1.2.3.4 (Server)	00:00:04 (Egress Router)
3	1.2.3.4 (Server)	10.3.0.101 (SNAT)	00:00:03 (SSLO Ext)
4	1.2.3.4 (Server)	10.1.0.1 (Client)	00:00:01 (Internal Router)

### Requirements:

<b>Inside next hop router</b>	Configured with SSLO as the default gateway
<b>Outside next hop router</b>	Configured with SSLO as the path for traffic destined for internal addresses. (unless next hop is selected)
<b>SSLO</b>	Deployed Network = L3 Network
<b>Clients</b>	Must trust the CA Certificate configured on SSLO

## Layer 3, Outbound explicit proxy

This topology provides an explicit outbound, or forward proxy solution to monitor traffic from internal users and systems going to external systems. In this topology each client application must be configured to send traffic to SSLO. This only works for HTTP traffic using a HTTP CONNECT header option. When SSLO is configured as an explicit proxy, and receives the HTTP CONNECT request, it will strip off the CONNECT header before forwarding the traffic to the service devices.



### Addressing Examples:

Step	Source IP	Dest IP	Dest MAC
1	10.1.0.1 (Client)	10.2.0.20 (Proxy Address) CONNECT: 1.2.3.4 (Server)	00:00:02 (SSLO Int)
2	10.3.0.101 (SNAT)	1.2.3.4 (Server)	00:00:04 (Egress Router)
3	1.2.3.4 (Server)	10.3.0.101 (SNAT)	00:00:03 (SSLO Ext)
4	1.2.3.4 (Server)	10.1.0.1 (Client)	00:00:01 (Internal Router)

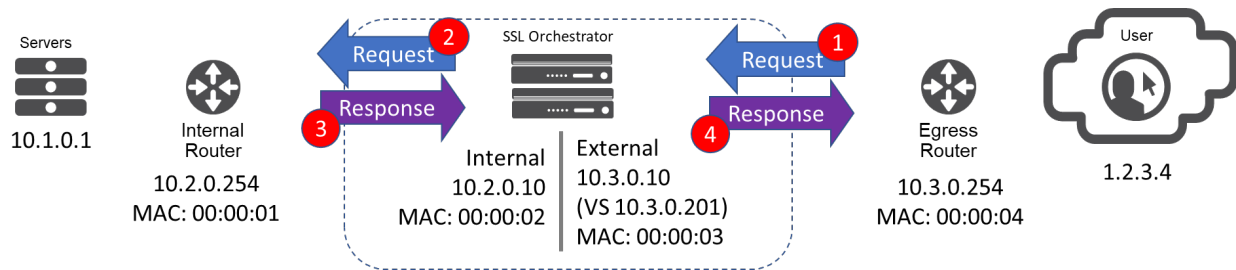
! SSLO does not support outbound explicit proxy connections over TLS. This is not the normal operational mode on most devices so should not be an issue.

### Requirements:

<b>Inside next hop router</b>	Configured with a route to SSLO
<b>Outside next hop router</b>	Configured with a route to SSLO
<b>SSLO</b>	Deployed Network = L3 Network
<b>Clients</b>	Configured with SSLO as the proxy address

## Layer 3, Inbound reverse proxy

This topology provides a transparent inbound, or reverse proxy solution to monitor traffic from external users and systems going to internal systems.



### Addressing Examples:

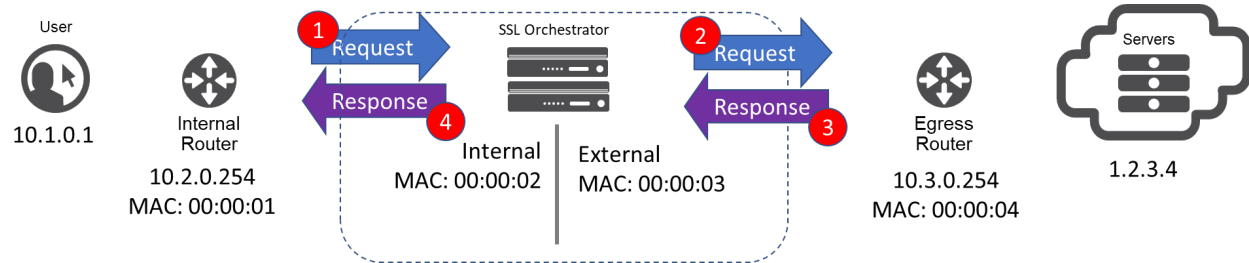
Step	Source IP	Dest IP	Dest MAC
1	1.2.3.4 (Client)	10.3.0.201 (SSLO)	00:00:03 (SSLO Ext)
2	1.2.3.4 (Client)	10.1.0.1 (Server)	00:00:01 (Internal Router)
3	10.1.0.1 (Server)	1.2.3.4 (Client)	00:00:02 (SSLO Int)
4	10.3.0.201 (SSLO)	1.2.3.4 (Client)	00:00:04 (Egress Router)

### Network Requirements:

<b>Inside next hop router</b>	Configured with a route to SSLO
<b>Outside next hop router</b>	Configured with the SSLO as the path for traffic destined for internal addresses.
<b>SSLO</b>	Deployed Network = L3 Network  (To use Layer 3 and HTTPS services on an inbound connection, see Configuring Layer 3 and HTTPS services for inbound traffic)
<b>External Clients</b>	No special configuration

## Layer 2, Outbound Transparent Proxy

This topology provides a transparent outbound, or forward proxy solution to monitor traffic from internal users and systems going to external systems without having to modify the customers routing environment.



### Addressing Examples:

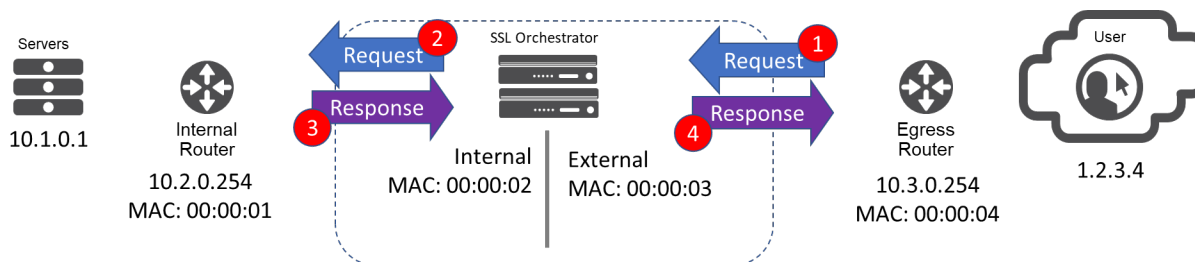
Step	Source IP	Dest IP	Dest MAC
1	10.1.0.1 (Client)	1.2.3.4 (Server)	00:00:04 (Egress Router)
2	10.1.0.1 (Client)	1.2.3.4 (Server)	00:00:04 (Egress Router)
3	1.2.3.4 (Server)	10.1.0.1 (Client)	00:00:01 (Internal Router)
4	1.2.3.4 (Server)	10.1.0.1 (Client)	00:00:01 (Internal Router)

### Network Requirements:

<b>Inside attached switch</b>	Configured with a VLAN that will force traffic to pass through the SSLO in order to get to the outside next hop router.
<b>Outside attached switch</b>	Configured with a VLAN that will force traffic to pass through the SSLO in order to get to the inside next hop router.
<b>SSLO</b>	Deployed Network = L2 Network
<b>Clients</b>	Must trust the CA Certificate configured on SSLO

## Layer 2, Inbound Transparent Proxy

This topology provides a transparent inbound, or reverse proxy solution to monitor traffic from external users and systems going to internal systems without having to modify the customers routing environment.



### Addressing Examples:

Step	Source IP	Dest IP	Dest MAC
1	1.2.3.4 (Client)	10.1.0.1 (Server)	00:00:01 (Internal Router)
2	1.2.3.4 (Client)	10.1.0.1 (Server)	00:00:01 (Internal Router)
3	10.1.0.1 (Server)	1.2.3.4 (Client)	00:00:04 (Egress Router)
4	10.1.0.1 (Server)	1.2.3.4 (Client)	00:00:04 (Egress Router)

### Network Requirements:

<b>Inside attached switch</b>	Configured with a VLAN that will force traffic to pass through the SSLO in order to get to the outside next hop router.
<b>Outside attached switch</b>	Configured with a VLAN that will force traffic to pass through the SSLO in order to get to the inside next hop router.
<b>SSLO</b>	Deployed Network = L2 Network To use Layer 3 and HTTPS services on an inbound connection, see <div>! Configuring Layer 3 and HTTPS services for inbound traffic</div>
<b>External Clients</b>	No special configuration

By default, HTTP and Layer 3 Services will not work with inbound workflows. (See

! Configuring Layer 3 and HTTPS services for inbound traffic)

## Scaling Options

SSL Orchestrator implementations can be scaled in various ways if additional performance or availability is needed beyond what one system is capable of.

Note that in many cases you can also combine these options.

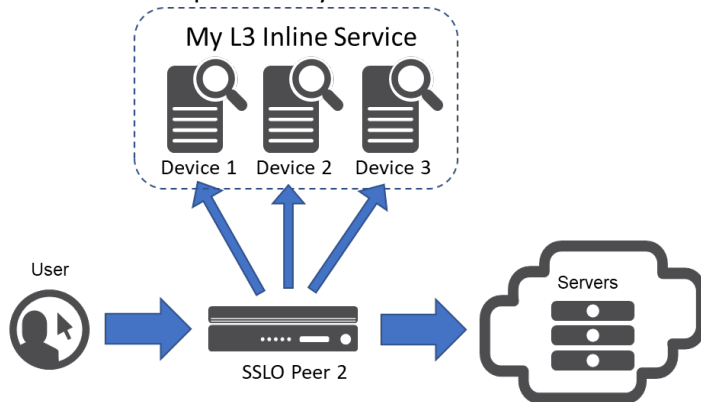
### Increase the size of the SSLO hardware

This is the simplest, but it's worth noting as an option. Since SSLO Systems are BIG-IP systems, the hardware may be able to be repurposed elsewhere and larger systems can be purchased.

! Note that at this time SSLO is not available for the VIPRION platforms.

### Add more security devices to each service

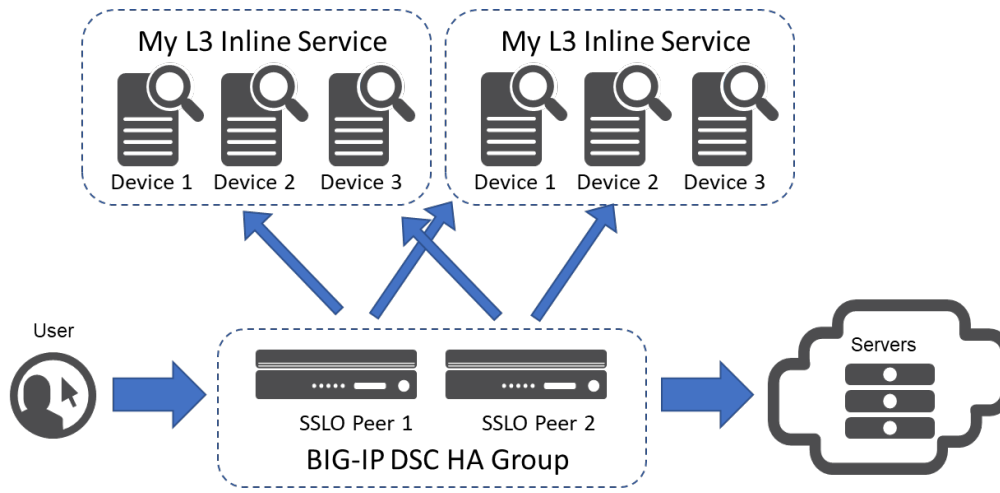
You can add multiple security devices to most of the services.



<b>Provides</b>	Additional capacity for each of the services
<b>Does not Provide</b>	Any overall additional capacity
<b>Use When</b>	The security services are reaching capacity, but the SSLO system still has headroom

## Implement Device Service Groups for SSLO System

This solution is most commonly used to provide high availability for SSL Orchestrator.



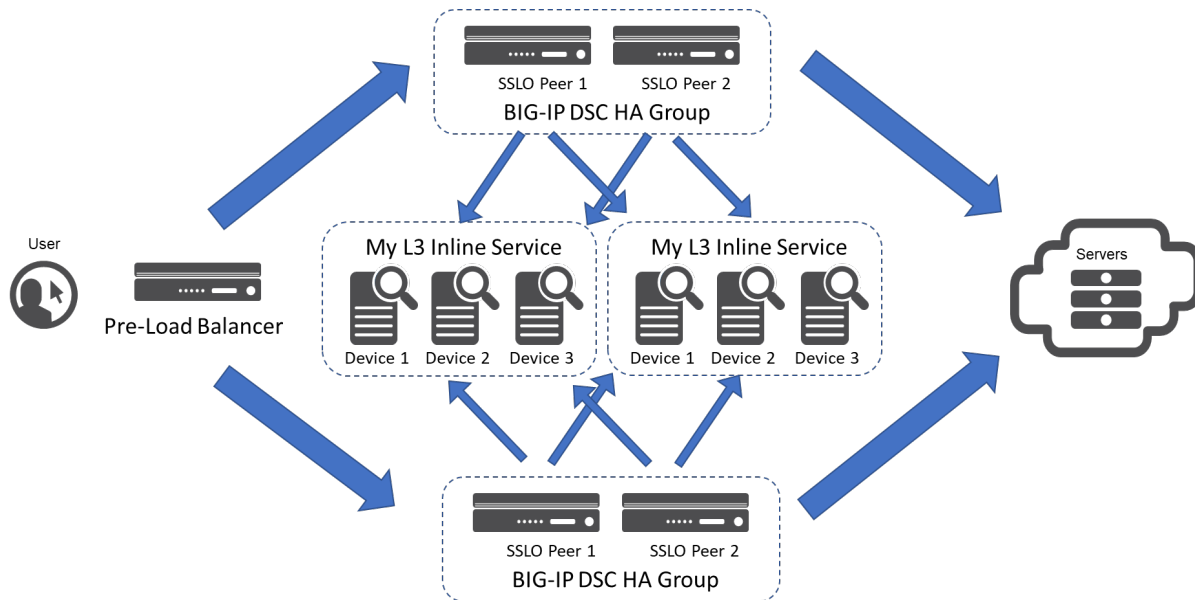
<b>Provides</b>	High Availability
<b>Does not Provide</b>	Any Additional capacity
<b>Use When</b>	SSLO is a mission critical piece of the infrastructure.

- ! When implementing HA, all services MUST use the same physical interfaces and VLAN IDs.
- ! When implementing DSC, always use manual config sync instead of automatic.
- ! When implementing DSC, never use a clone of the primary system for the second one. Always install from scratch. (using a clone will duplicate several unique system identifiers which will cause the sync process to fail).
- ! SSLO only supports two devices in an DSC at this point.



## Implement a load balancer in front of multiple SSL Orchestrator implementations.

This solution is used when large scale implementations are required. Note that due to the added complexity of this option, it should only be used when more performance is required than any given platform can provide.



<b>Provides</b>	Additional overall capacity
<b>Does not Provide</b>	Any additional capacity for the individual services
<b>Use When</b>	More capacity is required than a given SSLO platform can provide.

Note that HTTP and Layer 3 services do not support this approach out of the box and must be modified to work. (See

! Configuring Layer 3 and HTTPS services for inbound traffic)

# Service Architectures

The core of SSL Orchestrator are the services, services define the specific security devices that traffic is sent through.

SSL Orchestrator can support a number of different types of service types. Depending on the method that the service device uses to capture data.

The types of service devices are:

- Layer 2 Inline Devices
- Layer 3 Inline Devices
- Inline HTTP Proxy (Transparent or Explicit)
- ICAP Device
- TAP Device

In SSLO, the devices are defined as part of a service. Most services allow for multiple devices to be defined, allowing for scalability and availability.

! For services that support multiple devices, F5 has tested up to 8 devices. It is possible to add more than 8 devices to a service, but this has not been tested and should be done at your own risk.

## Services Security Best Practices

### Services should be near SSLO in protected address spaces

SSLO is almost never first in an enterprise architecture, so other security devices (proxies, firewalls, IPSs, sandboxes, etc.) are already there. If you ask any network and/or security admin, 99% of the time they have no interest in moving those security devices or re-architecting anything else when SSLO is introduced.

However, remember that the traffic being sent to and from the security devices is unencrypted, so sending traffic across an existing enterprise network to some security device is also sending passwords, credit card numbers, and lots of other PII data, across an uncontrolled span of network where any connected device can see it.

It is therefore brutally important that customers understand this, and that they do move those security devices to networks that are behind and protected by SSLO. In a perfect situation, no traffic should be able to reach these security devices except through SSLO. As frightening as the possibility of a data exposure is, there are going to be customers that ignore the warnings. You should therefore do your best to convey this security best practice to customers as their trusted advisor.

This is the reason for the “Auto-Manage” field in the configuration for the services. It creates non-overlapping, internal, non-routable address spaces for each service and encourages customers to protect them.

If disabling this, and changing the inline service IP spaces, take care to protect these.

### Minimize out-of-band exposure to inline security services.

These services now see decrypted traffic, so should be afforded more security than they may have had before. If traffic can flow in and out of a security service around SSLO, a compromise of this device could expose all of the decrypted data. Best practice is to only allow access to a security device through the F5.

If a service needs external access to remote resources, create an outbound interception rules (See

! Configuring external communications for services)

## Layer 2 Inline Service

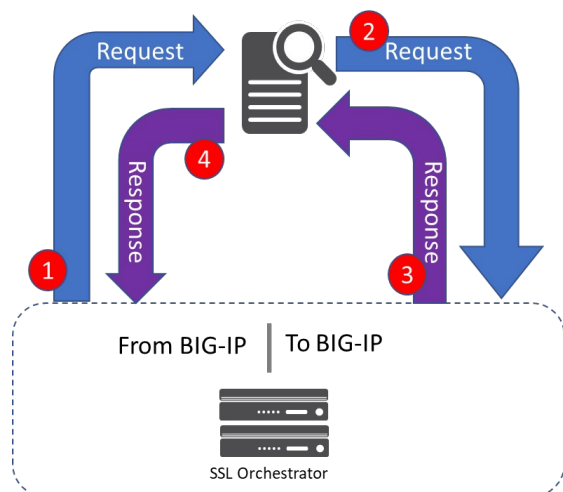
For configuration Reference see SSL Orchestrator v4.0 Reference Guide

A layer 2 inline device acts as a layer 2 switch and will forward traffic with the same broadcast domain. This type of device does not need to be addressed and will simply take traffic it receives, process it, and send it out the other side.

### Example of L2 Inline Services:

- FireEye NX
- Cisco Sourcefire
- Cisco Firepower
- Palo Alto Wildfire (in L2 mode)
- Tipping Point
- Gigamon network packet broker
- Ixia network packet broker
- Symantec DLP in L2 mode
- Any generic L2 IPS

### Network data flow



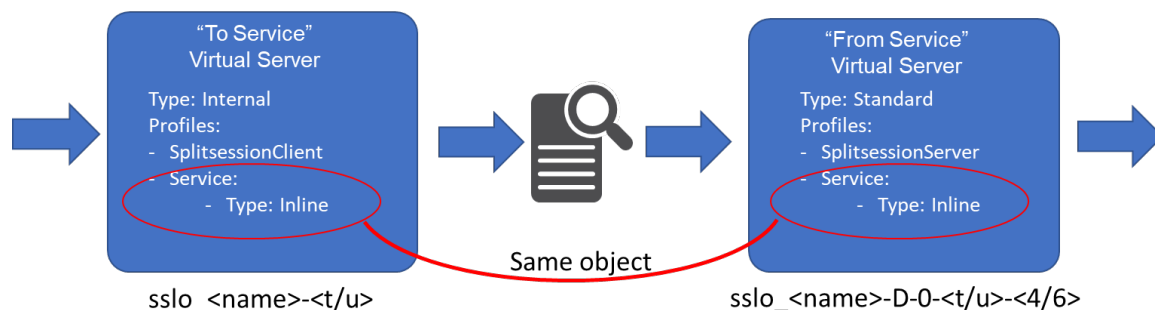
Step	Source IP	Dest IP	Dest MAC
1	Client	Server	SSLO MAC (to BIG-IP)
2	Client	Server	SSLO MAC (to BIG-IP)
3	Server	Client	SSLO MAC (From BIG-IP)
4	Server	Client	SSLO MAC (From BIG-IP)

## Requirements

<b>Network Requirements and Device Setup</b>	<p>This type of device requires two distinct VLANs, while the traffic for these will be considered in the same broadcast domain, they must not be linked in order to allow the security device to forward traffic.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>! These must be pre-created when implementing SSLO on a vCMP client</p> </div>
<b>Network Addressing</b>	<p>Internal Network Ranges used (next available selected)</p> <p>IPv4: 198.19.32.0/27 through 198.19.63.0/27</p> <p>IPv6: 2001:0200:0:0200::/123 through 2001:0200:0:02ff::/123</p> <p>Internally, the system will select address in the network selected. These are used for monitoring as well as other operations and are not configured on the service device..</p>
<b>Setup Information Needed</b>	<ul style="list-style-type: none"> <li>- VLAN connecting to and from the device.</li> <li>- If the device supports IPv6</li> </ul>
<b>Scalability</b>	<p>Administrators can define up to 8 devices in a single L2 inline service in SSLO, for each device, a weight can be defined which is used in load balancing decisions. [See BIG-IP load balancing for Ratio (Member)]</p>
<b>Monitoring</b>	<p>All devices are monitored using a gateway monitor.</p>
<b>Limitations</b>	<p>Port remapping only supported for outbound traffic by default. (See Remap ports for inbound traffic)</p>
<b>Options</b>	<p>Supports Port Remapping (Outbound only, see above)</p> <p>Supports Attaching an iRule</p>

## TMOS Objects:

Here is a high-level look at the configuration of the virtual server(s) created for this service type. Note this is representative, not complete.



## Layer 3 Inline Device

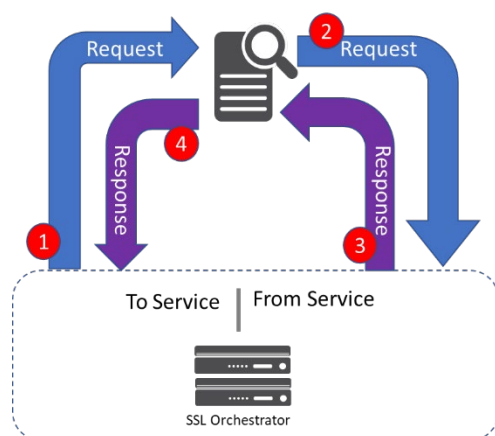
For configuration Reference see SSL Orchestrator v4.0 Reference Guide

A layer 3 inline device acts as a network router. Traffic is forwarded to it by the SSL Orchestrator by treating it as a next hop network device. The device will process the traffic and forward it back to the SSL Orchestrator as it's default gateway.

### Examples:

- Palo Alto Wildfire (in L3 mode)
- Any generic L3 IPS

### Network data flow



Step	Source IP	Dest IP	Dest MAC
1	Client	Server	Service MAC
2	Client	Server	SSLO MAC
3	Server	Client	Service MAC
4	Server	Client	SSLO MAC

### Requirements:

#### Network Requirements and Device Setup

The device should be setup with the service “From Service” address as it’s default gateway, and the “To Service” address as a next hop for internal addresses.

It should use an address that is in the range of addresses that have been defined in the service.

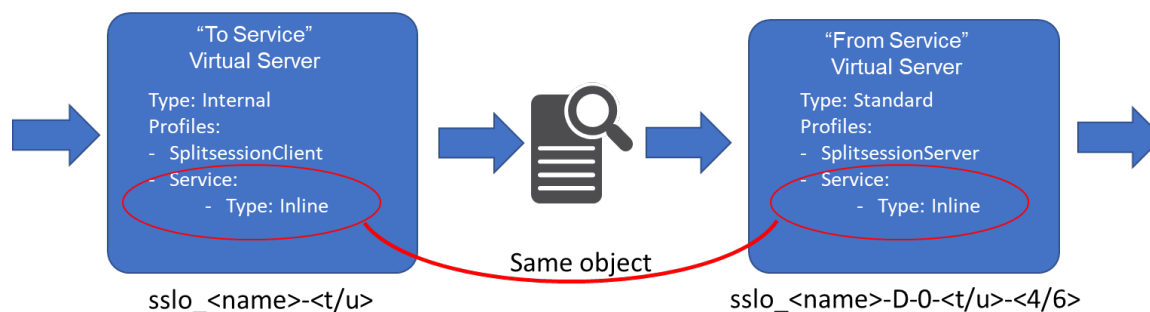
This uses two VLANs, one for traffic from the BIG-IP to the service devices, and another for traffic from the service device to the BIG-IP.

! These must be pre-created when implementing SSLO on a vCMP client

<b>Network Addressing</b>	<p>Internal Network Ranges used (next available selected)</p> <p>IPv4: 198.19.64.0/25 through 198.19.95.0/25</p> <p>IPv6: 2001:0200:0:0300::/120 through 2001:0200:0:03ff::/120</p> <p>Internally, the system will select three addresses in the network selected. The first and second addresses in the range are used for the floating and static IP, and the last address in the range is used for the inbound virtual server (to be set as the default gateway).</p>
<b>Setup Information Needed</b>	<ul style="list-style-type: none"> <li>- If the device is already addressed and you are unable or unwilling to change it, you will need the address.</li> <li>- If you are able to readdress it, you should hold off setting the address as you will not know it's address until it's configured in SSLO.</li> </ul>
<b>Scalability</b>	You can add up to 8 L3 Inline devices, which are load balanced using a predictive (node) method with each member having the same ratio.
<b>Monitoring</b>	All L3 devices are monitored using a gateway ICMP monitor.
<b>Limitations</b>	Traffic sent to an L3 device will ALWAYS be sent from the inside interface. This can cause problems for Layer 3 devices. (See
	<div>! Configuring Layer 3 and HTTPS services for inbound traffic)</div>
<b>Options</b>	<p>Supports Port Remapping</p> <p>Supports Attaching an iRule</p>

### TMOS Objects:

Here is a high-level look at the configuration of the virtual server(s) created for this service type. Note this is representative, not complete.



## Inline HTTP Proxy (Explicit / Transparent)

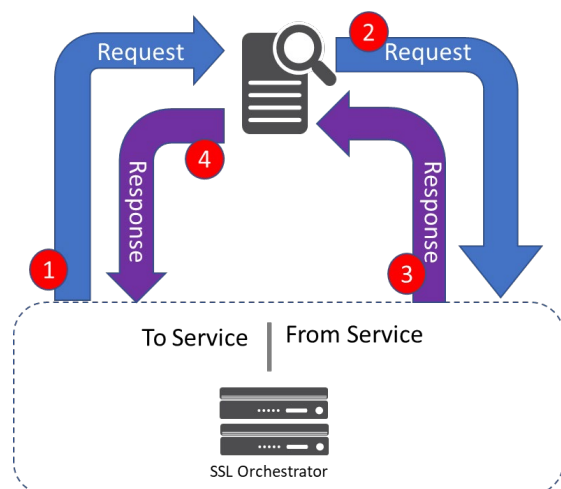
For configuration Reference see SSL Orchestrator v4.0 Reference Guide

Inline HTTP Proxy services will receive only HTTP traffic. If they are configured as an explicit proxy, a CONNECT header will be added to the HTTP header before sending it to the proxy.

### Examples:

- Blue Coat Proxies'
- Forcepoint gateway
- Cisco Web Security Appliance (WSA)
- McAfee Web Gateway (MWG)
- Squid

### Network data flow



Step	Source IP	Dest IP	Dest MAC
1	Client	Server (CONNECT: Service, if for explicit)	Service MAC
2	Client	Server	SSLO MAC
3	Server	Client	Service MAC
4	Server	Client	SSLO MAC

### Requirements:

#### Network Requirements and Device Setup

The device should be setup with the service “From Service” address as it’s default gateway, and the “To Service” address as a next hop for internal addresses.

It should use an address that is in the range of addresses that have been defined in the service.

If using Authentication Offload, the username (if known) will be passed to the device in the “X-Authenticated-User” header.

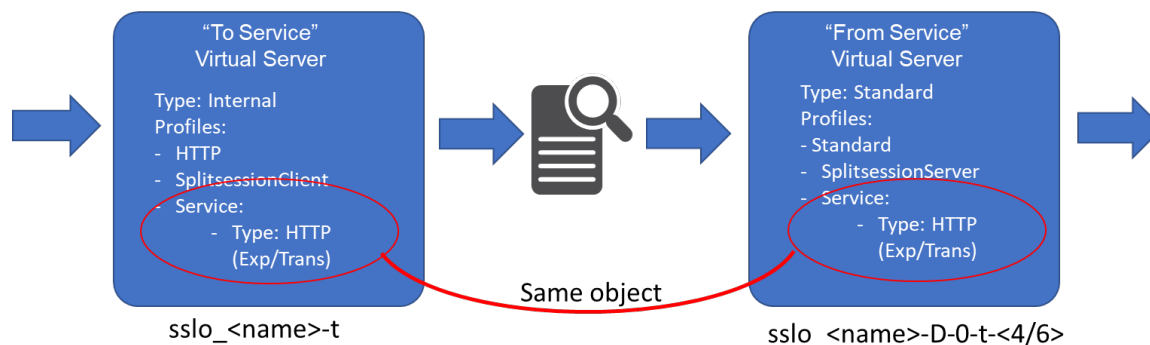
This uses two VLANs, one for traffic from the BIG-IP to the service devices, and another for traffic from the service device to the BIG-IP.

! These must be pre-created when implementing SSLO on a vCMP client

<b>Network Addressing</b>	<p>Internal Network Ranges used (next available selected)</p> <p>IPv4: 198.19.96.0/25 through 198.19.127.0/25</p> <p>IPv6: 2001:0200:0:0400::/120 through 2001:0200:0:04ff::/120</p> <p>Internally, the system will select three addresses in the network selected. The first and second addresses in the range are used for the floating and static IP, and the last address in the range is used for the inbound virtual server (to be set as the default gateway).</p>
<b>Setup Information Needed</b>	<ul style="list-style-type: none"> <li>- If the device is already addressed and you are unable or unwilling to change it, you will need the address.</li> <li>- If you are able to readdress it, you should hold off setting the address as you will not know it's address until it's configured in SSLO.</li> <li>- The mode of the device (Transparent or Explicit Proxy)</li> </ul>
<b>Scalability</b>	You can add up to 8 devices which are load balanced using a Predictive (Node) method with each member having the same ratio.
<b>Monitoring</b>	Each member is monitored using a gateway ICMP monitor.
<b>Limitations</b>	<p>Is only used for HTTP(s) conversations</p> <p>Traffic sent to an HTTP device will ALWAYS be sent from the inside interface</p> <p>(See</p> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <p><b>!</b> Configuring Layer 3 and HTTPS services for inbound traffic)</p> </div>
<b>Options</b>	<p>Supports Port Remapping (for Transparent)</p> <p>Supports Attaching an iRule</p> <p>Supports Authentication Offload</p>

## TMOS Objects:

Here is a high-level look at the configuration of the virtual server(s) created for this service type. Note this is representative, not complete.





# ICAP Device

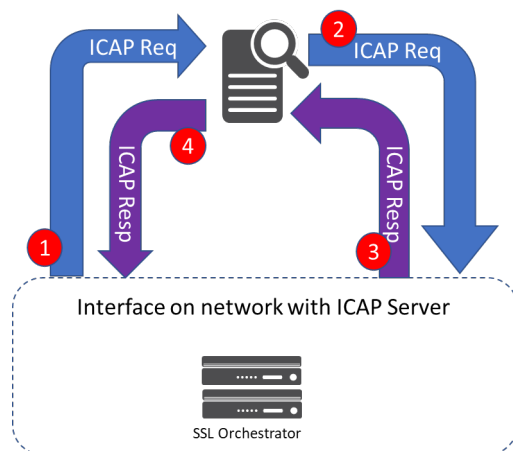
For configuration Reference see SSL Orchestrator v4.0 Reference Guide

An ICAP (Internet Content Adaptation Protocol) device is sent the HTTP header and a portion of the content from a request or response. The device will then decide to let it go through, block it, or replace it with something else.

## Examples:

- Symantec DLP
- McAfee DLP
- Blue Coat CAS
- Codegreen DLP
- C-ICAP (with Squid and Clamav)

## Network data flow



Step	Source IP	Dest IP	Dest MAC
1	SSLO	ICAP Server	ICAP Server MAC
2	ICAP Server	SSLO	SSLO MAC
3	SSLO	ICAP Server	ICAP Server MAC
4	ICAP Server	SSLO	SSLO MAC

## Requirements:

<b>Network Requirements and Device Setup</b>	This service does not define VLANs or address ranges. Instead traffic is sent to a defined address using the normal BIG-IP routing controls. The ICAP device must have a routed path back to the BIG-IP in order to send responses back.
<b>Network Addressing</b>	The ICAP service does not generate internal addresses, it will route the traffic to any interface using the system routing table.
<b>Setup Information Needed</b>	<ul style="list-style-type: none"><li>- The address of the ICAP server(s)</li><li>- The port number being used (defaults to 3128)</li><li>- The VLAN that will lead to the ICAP server(s)</li><li>- Any specific headers that are needed</li><li>- The URI for requests and responses</li></ul>

<b>Scalability</b>	You can add up to 8 devices which are load balanced using a Predictive (Node) method with each member having the same ratio.
<b>Monitoring</b>	Each member is monitored using a TCP monitor using the defined ICAP port.
<b>Limitations</b>	Is only used for HTTP(s) conversations.
<b>Options</b>	Supports ICAP Header modification Supports setting the size of the content forwarded to the ICAP server. Supports HTTP version selection Supports defining an ICAP policy in LTM CPM.

### ICAP Policy Information:

An ICAP policy is an LTM policy that allows bypassing the ICAP service for certain traffic. This is achieved by disabling the “request-adapt” or “response-adapt” option (which is the BIG-IP process that handles ICAP).

In order to use an ICAP policy, the policy must meet the following criteria:

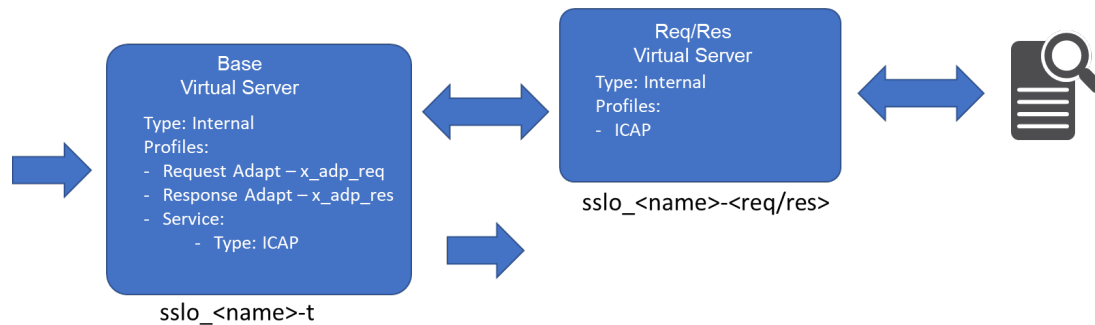
- Disabling ICAP for requests:
  - The condition must be an HTTP condition (should start with HTTP)
  - The condition must be one that fits a request (so, not “HTTP set cookie” or “HTTP status” for example)
  - The condition must be evaluated at request time
  - The action must be “disable / request-adapt” at “request” time.
- Disabling ICAP for responses at request time:
  - Same as above, however, the action must be “disable / response-adapt” at “response” time.
- Disabling ICAP for responses at response time:
  - The condition must be an HTTP condition (should start with HTTP)
  - The condition must be one that fits a response (so, not “HTTP user agent” for example)
  - The condition must be evaluated at response time
  - The action must be “disable / response-adapt” at “response” time.

Uses for this could include:

- Disabling ICAP response processing for authentication pages (status code 407)
- Disabling ICAP requests processing for uploads to certain sites
- Disabling ICAP response processing for downloads from certain sites

## TMOS Objects:

Here is a high-level look at the configuration of the virtual server(s) created for this service type. Note this is representative, not complete.



## TAP Device

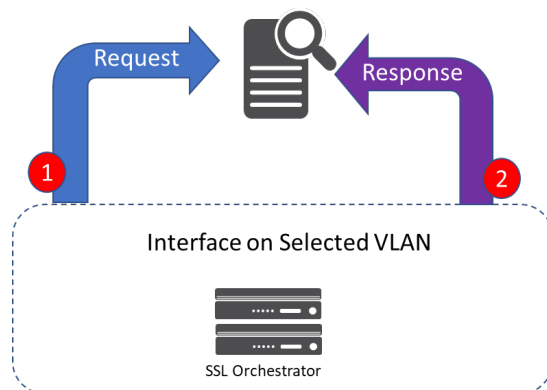
For configuration Reference see SSL Orchestrator v4.0 Reference Guide

A tap service device will receive a copy of the traffic sent to it. It has no ability to modify the traffic and is not inline, so a device that is not operating will not impact traffic going through the system.

### Examples:

- Cisco Sourcefire
- Cisco Firepower
- Snort
- Trend Micro Deep Discovery Inspector / Analyzer

### Network data flow



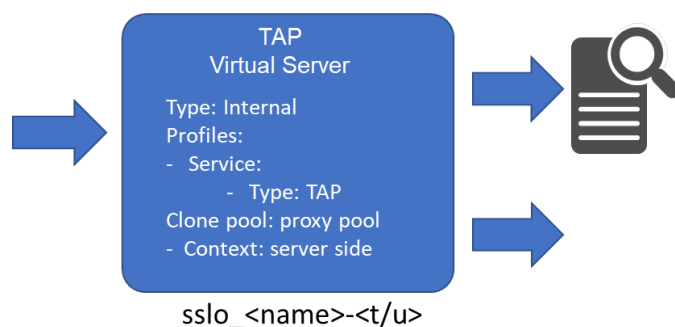
Step	Source IP	Dest IP	Dest MAC
1	Client	Server	TAP Service MAC
2	Server	Client	TAP Service MAC

## Requirements:

<b>Network Requirements and Device Setup</b>	Traffic that is sent to a TAP port is transmitted on the VLAN that was defined in the setup. It is recommended that the TAP device be directly connected to the BIG-IP to this port. However as long as it is within the same broadcast domain it should be able to receive traffic (*assuming that all in between switches are correctly forwarding traffic for the specified MAC address).
<b>Network Addressing</b>	Internal Network Ranges used (next available selected) IPv4: 198.19.0.0/30 through 198.19.31.0/30 IPv6: 2001:0200:0:0100::/124 through 2001:0200:0:01ff::/124  Internally, the system will select one address in the network selected. The IP address is for internal use to route the packet through ARP/NDP.
<b>Setup Information Needed</b>	<ul style="list-style-type: none"><li>- The MAC address of the interface that should receive the traffic on the device.</li><li>- The Interface and VLAN that should be used to send traffic to this device</li></ul>
<b>Scalability</b>	You can only define one destination MAC address, however, you could setup multiple devices on that VLAN to monitor this traffic.
<b>Monitoring</b>	Devices are not monitored by default (See Adding a monitor to a TAP service)
<b>Limitations</b>	Note that the destination MAC address is replaced with the device's MAC address, so the receiving device should not use destination MAC for analysis.
<b>Options</b>	Supports port remap

## TMOS Objects:

Here is a high-level look at the configuration of the virtual server(s) created for this service type. Note this is representative, not complete.



# Policy Architecture

The policy architecture of SSLO revolved around the interception rules and policies. Much of this is built by default when creating the default rules from the Interception rules page, but for troubleshooting or advanced configuration, you may need to create some of these objects on their own and link them together.

## Overall policy object relationships

SSL Orchestrator uses a number of objects to define how traffic is handled, these minimally include;

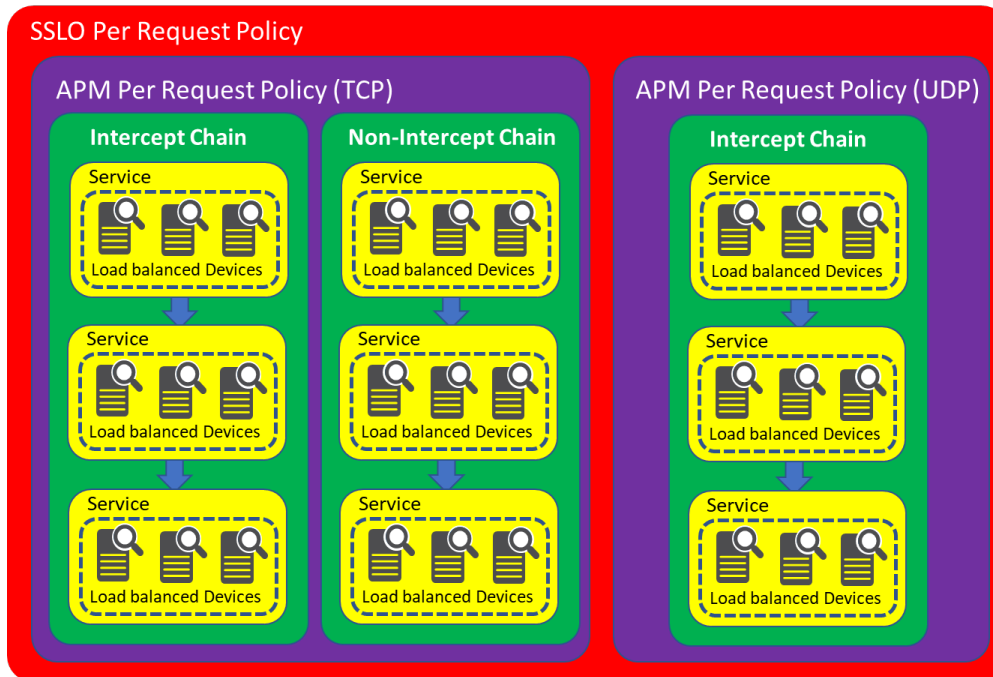
<b>SSL Settings Group</b>	Defines how SSL traffic will be handled including defining certificates and SSL settings.
<b>Service Chain</b>	Determines the services to use, and the order in which to use them. (contains services)
<b>APM Per-Request Policy</b>	Defines how the traffic will be processed including branching and filtering. (contains one or more service chains)
<b>APM Per-Session Policy (Access Profile)</b>	Defines once per session actions such as authentication.
<b>Rule</b>	Wraps SSL Settings, Per-Request Policies, and Access Profiles together. May contain one SSL Settings Group, one APM per-request policy, and one Access Profile, along with other settings.

## SSL Settings Groups

These objects will define the BIG-IP SSL client and server profiles. They identify the specific certificates that will be used as well as the ciphers allowed. You can have a different SSL Settings Group object for each SSLO Rule in the system.

## SSLO Per-Request Policy

The Per-Request Policy in the SSL Orchestrator menu is actually a container object for two APM per-request policy objects (also called Per Request Policies), one for UDP traffic with one chain and one for TCP traffic with two. In this guide, we will call the container object the SSLO Per-Request Policy to differentiate them from the APM Per-Request Policies.



Per-Request policies for SSL Orchestrator rules must be created within the SSL Orchestrator menu section so that the two APM objects are correctly created.

In SSL Orchestrator, you use the SSLO Per-Request Policy option to create these. When you link these to the rules you will actually be referencing the APM Per-Request Policies. So, think of the SSLO Per-Request Policy as a wizard to help create the APM Per-Request Policies. It is not used on its own elsewhere.

When you create a per-request policy in SSL Orchestrator, you are asked to define service chains. These each define zero or more services that traffic will flow through, and the order in which they are used.

! If you do not define a service in a chain, traffic will pass through the system un-monitored. This can be a good way of troubleshooting the system without running traffic through the third-party systems.

You have the ability to create three service chains when creating an SSLO Per-Request Policy.

## TCP Chains

These chains are used in the default TCP APM Per-Request policy.

### TCP Intercept Chain

For TCP traffic to be decrypted and passed through the defined and ordered services.

### TCP Non-Intercept Chain

For TCP traffic that will be passed through the defined and ordered services without being decrypted

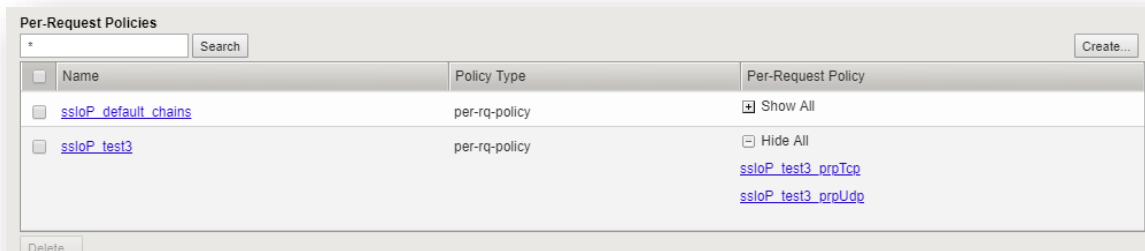
**!** You can add ICAP and HTTP services to this chain, however, traffic will not be forwarded to them.

## UDP Service Chain

This chain is created and used in the default UDP APM Per-Request Policy. You will not see any HTTP specific options in this service chain such as HTTP and ICAP services.

## APM Per-Request Policies

Creating the Per-Request Policies in the SSLO Per-Request Policy wizard will result in two APM per-request policy objects called “ssloP\_<name>\_prpTcp” and “ssloP\_<name>\_prpUdp”. These can be seen by clicking on the “Show All” button under the Per-Request Policy column in the Per-Request Policies list.



If you click on the APM per-request policy link you will open the Visual Policy Editor (VPE) and can modify the actual policy.

See the Default APM Per-Request Policy section for more information on these.

## Access Profile (Per-Session Policy)

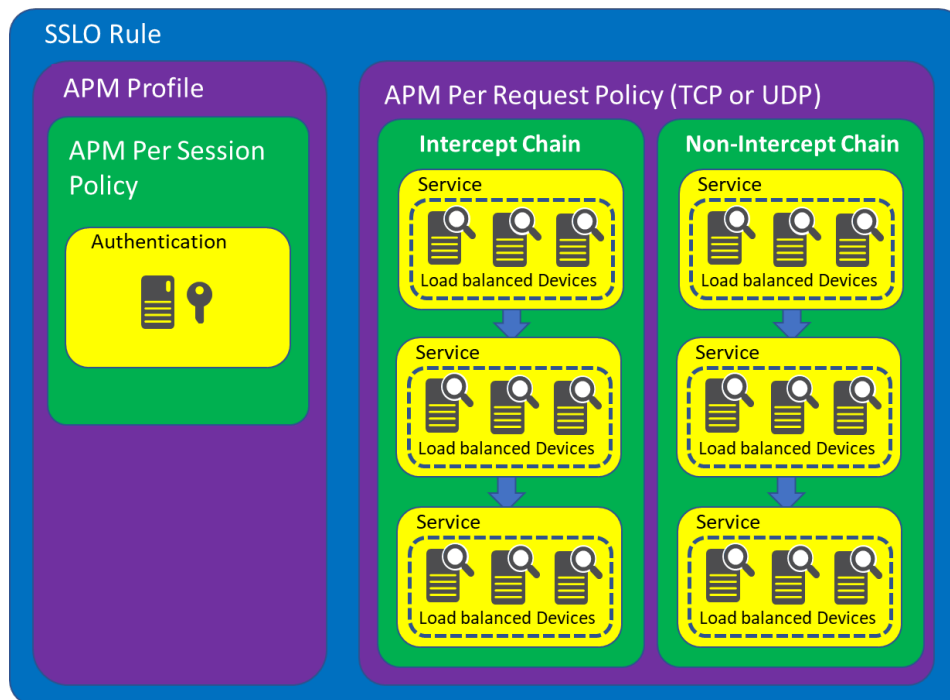
SSLO can use the following types of access profiles on a per-session basis. In other words, these are activated only once per user session, instead of once per request. These would be for things like authentication.

- SSL Orchestrator (this is the default and performs no actions per session).
- SWG – Explicit (only available if APM is licensed)

These are selected when creating custom inbound and outbound rules.

## Interception Rules

Interception Rules are the object that ties all of the various policy objects together. A rule defines a specific type of traffic to listen for and how to handle it when it arrives. The rules will also link to an APM session profile (optionally containing a per-session policy) and APM Per-Request Policy (NOT an SSLO Per-Request Policy).



Once you create the rules, either with the Create Default Rules or in adding a custom inbound or outbound rule, the system will setup a virtual server listener for the rules.



## Outbound Rules

Outbound rules can be explicit or transparent and will operate as a forward proxy.

If you create an explicit outbound rule (by checking the “explicit” checkbox), this rule is processed first, then that rule sends the traffic to the “-in-<4/6>” VIP for processing after stripping off the HTTP CONNECT header.

Because of this, on the explicit rule, no SSL settings are set nor are any policy and profile settings available, these are handled by the main listening VIP.

**!** By default, there can only be ONE outbound explicit listener. (See [Create additional outbound explicit proxy rules](#))

SSLO does not apply IP address persistence to outbound traffic, in certain circumstances this can cause connectivity problems. (See [IP Address Persistence](#))

**!** Provide IP address persistence for outbound traffic)

## Inbound Rules

Inbound rules must be transparent and operate as a reverse proxy.

Inbound rules operate the same as outbound rules, with traffic flowing through the services in the normal defined order. (SSLO does NOT reverse the order of the services for inbound rules).

There are some known issues around inbound rules:

Inbound rules must have SNAT disabled (See [SNAT](#))

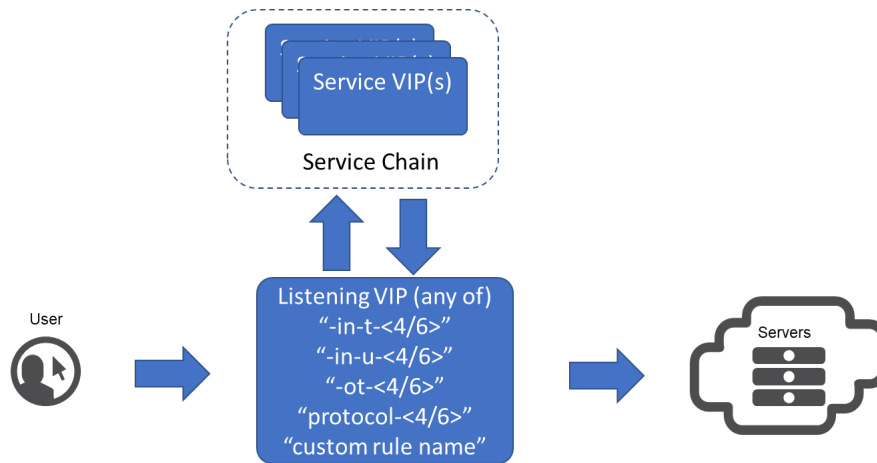
**!** Configuring Inbound Services when SNAT is used)

Inbound rules may not work with Inline HTTP or Layer 3 Services (See [Layer 3 Services](#))

**!** Configuring Layer 3 and HTTPS services for inbound traffic)

## Transparent Proxy Rule Data Flow

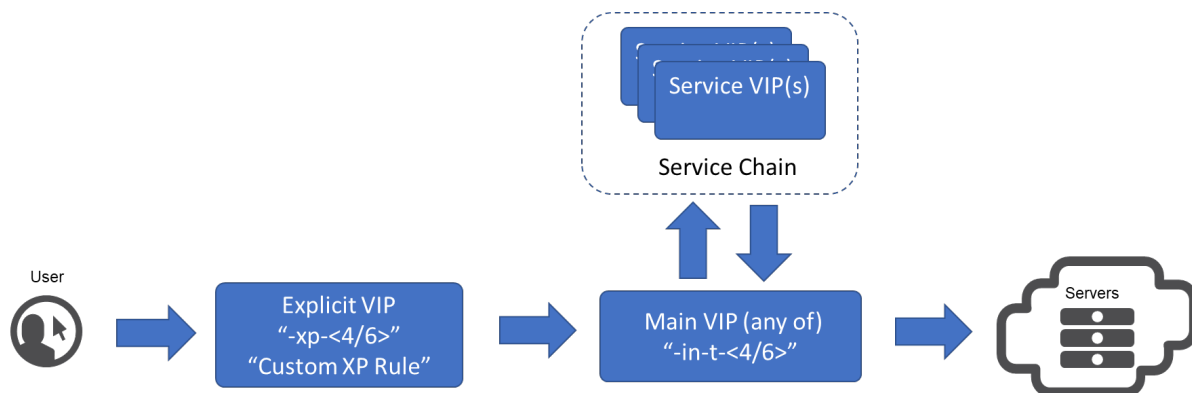
Traffic will enter the system on the listening VIP, then be passed to the various services (see: [Service Architectures](#)) before exiting to the servers.



*All non-service VIP names start with "sslo\_<configuration name>"*

## Explicit Proxy Rule Data Flow

Traffic will enter the system on the on the "ex" (explicit proxy) virtual server, which strips off the CONNECT header and passes it to the main listening virtual server, then be passed to the various services (see: [Service Architectures](#)) before exiting to the servers.



*All non-service VIP names start with "sslo\_<configuration name>"*

## Default APM Per-Request Policies

---

This section provides a reference for the VPE policy that is created when creating a per-request policy. This covers the use and default configuration of the created macros and agents in the policy. For more information on the specific agent fields and options, see the SSL Orchestrator v4.0 Reference Guide.

There are two different default APM Per-Request Policies created;

- /Common/ssloP\_test3.app/ssloP\_test3\_prpTcp (for TCP traffic)
- /Common/ssloP\_test3.app/ssloP\_test3\_prpUdp (for UDP traffic)

In addition to the base policy, the TCP policy contains the following macros:

- Categorization – Used to initially categorize traffic
- IP Policy – \*Not used by default
- Proxy Chaining(Connect) - \*Not used by default
- Proxy Chaining(URI Rewrite) – \*Not used by default
- SSL Intercept Policy – Determines when to decrypt traffic
- Service Chain Intercepted – Created based on the services in the TCP Intercepted service chain.
- Service Chain Not Intercepted – Created based on the services in the TCP Non-Intercepted service chain.

In addition to the base policy, the UDP policy contains the following macros:

- IP Policy - \*Not used by default
- Service Chain – Created based on the services in the UDP service chain.

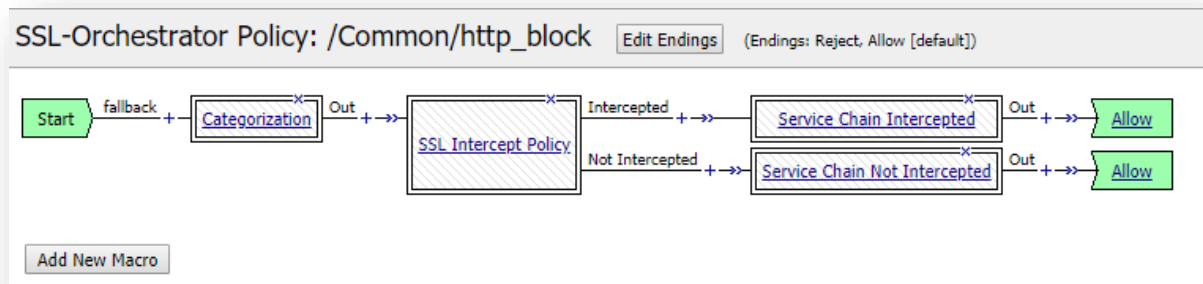
The following sections detail how these are implemented by default. (For more information on all of the specific agent fields and options, see the SSL Orchestrator v4.0 Reference Guide)

## Base Policy

The top item is the policy, which is the base item for the overall policy. This controls which macros are used in the policy, and the flow through them.

The base policy differs in the TCP and UDP policies. The individual macros used in each are detailed below.

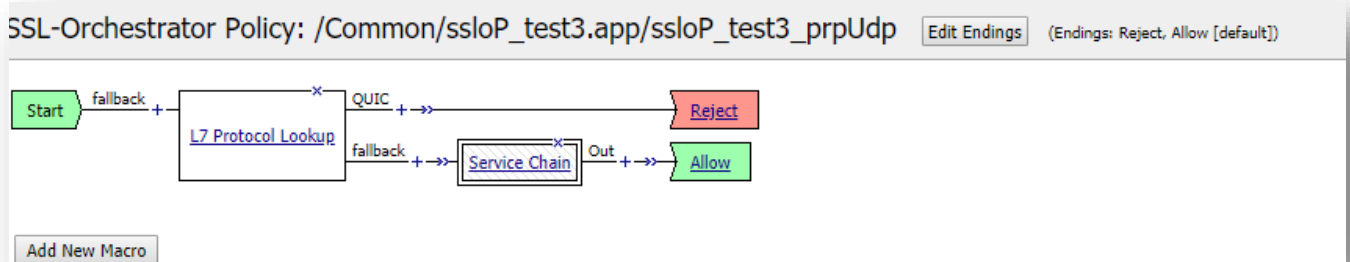
## TCP Base Policy



## Macros

<b>Categorization</b>	This macro is used to handle blocking of various types of traffic.
<b>SSL Intercept Policy</b>	This macro is normally used to determine if the traffic is decrypted and run through the service chain(s).
<b>Service Chain Intercepted</b>	This is the intercepted (decrypted) service chain macro.
<b>Service Chain Not Intercepted</b>	This is the not intercepted (encrypted) service chain macro

## UDP Base Policy



## Macros:

<b>Service Chain</b>	This is the service chain macro that routes traffic through the services.
----------------------	---

## Agents:

<b>L7 Protocol Lookup</b>	This step determines if the connection is using QUIC or not.
---------------------------	--

*Base: Empty*

### Branches:

QUIC	QUIC is found to be in use. (This will reject the connection).
Fallback	

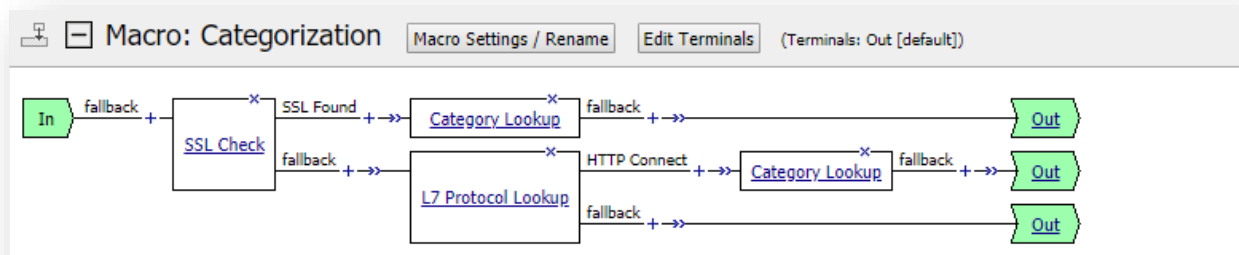
### Notes:

See above section on QUIC for more information.

## Categorization Macro

The Categorization macro handles checking for situations that should completely block traffic, as well as providing a place to do lookups and store data in variables if needed for future macros.

Most of these macros are initially empty and are intended as a place to provide custom branching.



## Agents:

<b>SSL Check</b>	The SSL Check step determines if TLS/SSL is being used for this connection.
------------------	---

*Base: Empty*

### Branches:

SSL Found	SSL is found to be in use.
Fallback	

### Notes:

One of the main reasons this is here is that for SSL connections, we want to do URL categorizations based on the SNI (if present) instead of simply an address or pulling the URL from the HTTP header.

---

**Category Lookup**

*Base: Category Lookup*

This step defines how URL Category lookup gets its information. By default, the system will look at the SNI (Server Name Indicator) in the SSL client hello. Since this is of course only present in SSL connections, we needed that previous “SSL Check” step to filter for them.

**Settings:**

Categorization Input	Use SNI in client Hello
Category Lookup Type	Process custom categories only
Reset on Failure	Enabled

**Branches:**

fallback	
----------	--

**Notes:**

You can add additional branch rules here based on DNS names.

! This object is incorrectly deployed in SSLO v4.0 and should be fixed after implementation. (See Setup without URL Filtering or SWG licensed)

---

**L7 Protocol Lookup**

*Base: Empty*

By default, this step will filter connections that are intended for an HTTP Explicit proxy (that have the “CONNECT” header). This allows us to use that header in URL categorization if present for non-TLS connections.

**Branches:**

HTTP Connect	If connection is aimed at an explicit HTTP proxy server
fallback	

---

**Category Lookup (from L7 Protocol Lookup)**

*Base: Category Lookup*

This step tells SSLO to do URL categorization based on the HTTP CONNECT header in the case of non-TLS traffic.

**Settings:**

Categorization Input	Use HTTP Connect Hostname
Category Lookup Type	Process custom categories only
Reset on Failure	Enabled

**Branches:**

Fallback	
----------	--

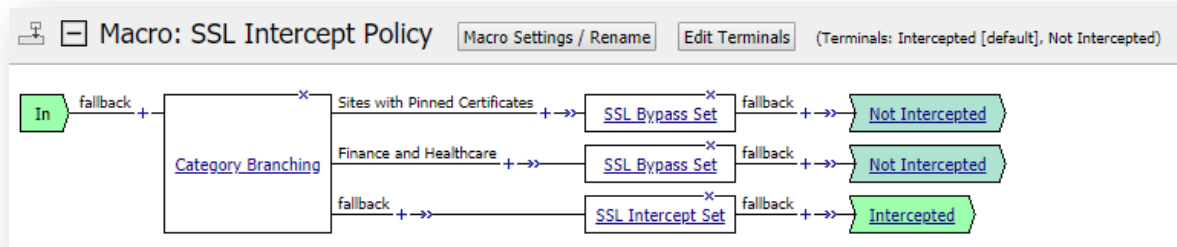
---

# SSL Intercept Policy Macro

The SSL Intercept Policy Macro determines which service chain the traffic will use, and if that traffic is decrypted or not when passing it through the service chain.

The default for this macro will send traffic from known sites that have pinned certificates and cause problems to the Non-Intercepted chain, in addition, it will send traffic categorized as “Finance and Healthcare” to the non-intercept site for privacy reasons. These can of course be changed as needed.

Remember to add additional terminals if you are going to be selecting between different chains.



## Agents

### Category Branching

Base: Empty

This is the step that can be used to select the connections that should follow different chains, and which ones should be decrypted or not.

### Branches:

Sites with Pinned Certificates	Sends URLs in the “Custom – Pinners” category to this branch
Finance and Healthcare	Sends URL’s in the “Financial and Data Services” OR “Health and Medicine categories” to this branch
Fallback	

### Notes:

You can add extra branches to this macro to select the data needed. If you do not have URL Filtering or SWG licensed and configured, the Finance and Healthcare branch will not be detected.

---

**SSL Intercept Set**

*Base: SSL Intercept Set*

This is the step that will actually decrypt the traffic before sending it to the chain.

**Settings:**

Agent	Intercept
-------	-----------

**Branches:**

fallback	
----------	--

**Notes:**

There is only one of these by default, but you should have one for each branch from the categorization step(s), and one before each termination point, each should either intercept (Decrypt) or bypass traffic.

You should NOT add extra branches to this step.

---

---

**SSL Bypass Set (x2)**

*Base: SSL Bypass Set*

This is the step that will send encrypted traffic through the Non-Intercept chain.

**Settings:**

Agent	Bypass
-------	--------

**Branches:**

fallback	
----------	--

**Notes:**

You should NOT add extra branches to this step.

---

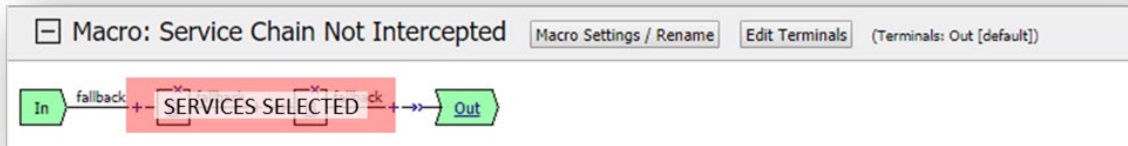
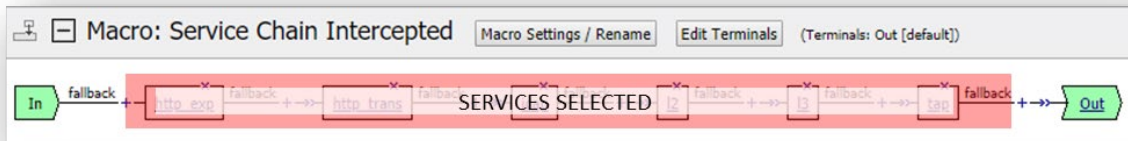


## Service Chain Intercepted Macro

---

This macro contains a list of the services that the traffic must traverse when following this chain.

Two of these are initially created based on the settings from the default rule creation.



### Agents

---

#### <Service Step>

Base: Service Connect

Each step uses the service connect agent to select the service that it should be using.

#### Settings:

Connector Profile	<selected service>
-------------------	--------------------

#### Branches:

fallback	
----------	--

#### Notes:

If these are created using the SSLO chain interface, they should be edited from there as well.

## IP Policy Macro (Unused)

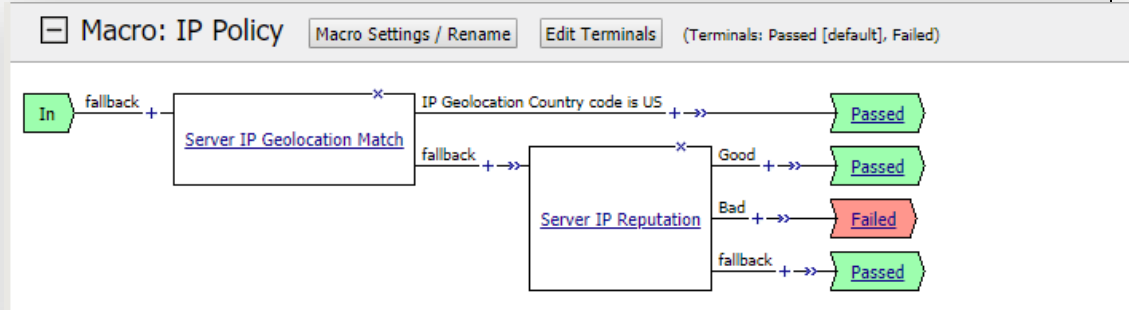
---

This macro can be used when you would like to block traffic using IP address information. This is an example policy macro and should be modified before use. The example approach of this macro is to first check for traffic destined for a US based server, and if the traffic is destined for a non-US server, check the reputation of the destination with the IP Reputation database.

! See Select a different intercept chain based on IP intelligence or geolocation

This macro would normally be inserted in the beginning of the policy to filter by IP.

! To enhance performance when using this macro, change the “One Shot Macro” option to “Yes” in the Macro Settings. See Select a different intercept chain based on IP intelligence or geolocation



!

**Agents:**

**Server IP Geolocation Mach**  
*Base: IP Geolocation Lookup*

This step will check, and block based on the geolocation of the server address.

**Settings:**

Input Source	Server
Reset of failure	Enabled

**Branches:**

IP Geolocation Country Code is US	Country code == US Goes directly to pass terminal
Fallback	Goes to the IP reputation lookup step.

**Notes:**

It is expected that the admin will modify this to fit their Geolocation needs.

! IP Geolocation databases must be installed and configured for this to work. If they are not, it will fail silently and will not pass traffic.

**Server IP Reputation**  
*Base: IP Reputation  
Lookup*

This step will perform an IP Reputation lookup agent based on the server IP address.

**Settings:**

Input Source	Server
Reset of failure	Enabled

**Branches:**

Good	No IP reputation
Bad	IP Reputation in list of problem addresses.
Fallback	

**Notes:**

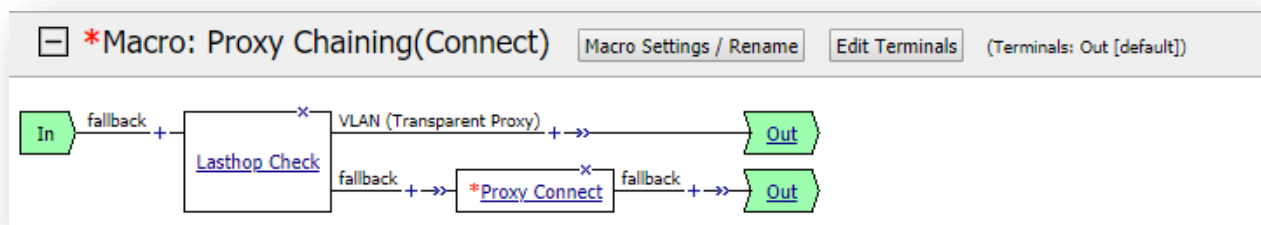
! For this to work, IP Intelligence must be configured, licensed, and working. If this is not setup, traffic will fail silently.

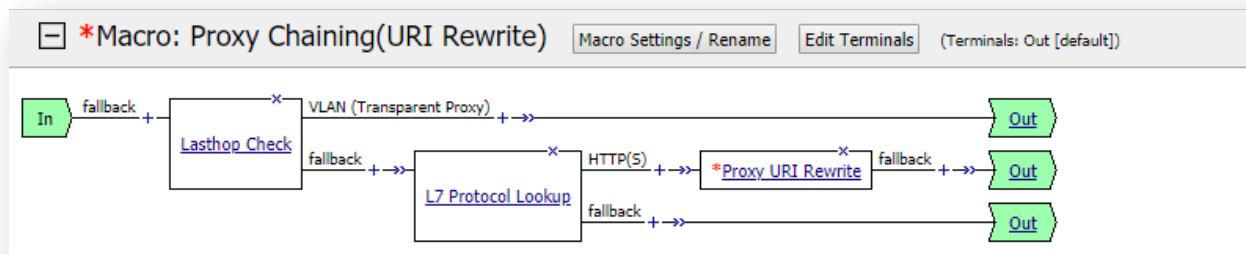
## Proxy Chaining (Connect / URI Rewrite) Macros (Unused)

The two proxy chaining macros (Connect and URI Rewrite) are used when you need to send traffic through an explicit HTTP proxy device after it leaves the SSLO system.

- These would be configured by adding them to the policy base. The “Proxy Chaining(Connect)” macro would go in the beginning of the policy, to make sure that all traffic is forwarded to the proxy, and the “Proxy Chaining(URI Rewrite)” macro goes at the end to actually re-write the URI. (See

Configuring SSLO to use an upstream explicit proxy)





## Agents

### Lasthop Check

Base: Empty

This detects if the traffic is coming from a Transparent or Explicit proxy rule.

#### Branches:

VLAN (Transparent Proxy)	Skips the proxy re-write step.
fallback	Goes to L7 Protocol Lookup when in the URI Rewrite macro, and direct to the Proxy Connect step otherwise.

#### Notes:

Since traffic from a transparent proxy does not need to be re-write, we will bypass the URI-rewrite section. Otherwise, we need to re-write the existing CONNECT header.

### L7 Protocol Lookup

Base: Empty

When in URI Rewrite macro, this checks to see if the traffic is HTTP or HTTPS, and if so, sends the traffic through the rewrite step.

#### Branches:

HTTP(s)	Checks for HTTP/HTTPS traffic, sends to the Proxy Rewrite step
fallback	Bypasses the proxy rewrite step

---

**Proxy Connect**  
*Base: Proxy Select*

Traffic passing through this step will automatically be passed to the defined pool of proxy servers. If they are explicit, a username and password can be provided, and HTTP header information can be updated.

**Settings:**

Pool	*Required <unset> Select the pool of proxy servers to send traffic to.
Upstream Proxy Mode	Explicit (see notes)
Username	<blank>
Password	<blank>

**Branches:**

fallback	
----------	--

**Notes:**

3. The Proxy Connect agent configuration must be changed for it to operate correctly in an upstream explicit proxy implementation (See

Configuring SSLO to use an upstream explicit proxy.)

---

**Proxy URI Rewrite**  
*Base: Proxy Select*

This agent will re-write any existing CONNECT headers to the correct ones for the upstream explicit proxy.

**Settings:**

Pool	<unset> Select the pool of proxy servers to send traffic to.
Upstream Proxy Mode	Explicit
Username	<blank>
Password	<blank>

**Branches:**

Fallback	
----------	--

---

# Troubleshooting

As with anything in the technology world, problems are usually inevitable and poorly timed. In the event that an SSL Orchestrator configuration is not behaving as expected, you may find the following troubleshooting suggestions useful. For additional suggestions, use the AskF5.com knowledgebase.

For the most part, SSL Orchestrator operates using mostly normal BIG-IP objects, so when troubleshooting and searching, make sure not to narrow your searches to only SSL Orchestrator problems.

The below items are intended to provide a starting point and overview of areas to look at in troubleshooting SSL Orchestrator, for more in-depth instructions on these points and other troubleshooting tips, please refer to the AskF5.com knowledgebase.

## Prior to troubleshooting

Prior to troubleshooting, you should take the following actions;

- Make sure to review this Architecture guide as well as any use cases that are similar to your deployment for known limitations and workarounds.
- Back up your configuration and insure your documentation is up to date.
- Document any system customizations that you have performed (including updating iRules and modifying BIG-IP objects) and back these out for testing.
- Upload a Qkview to iHealth (see [iHealth.f5.com](http://iHealth.f5.com) and [askF5.com](http://askF5.com)). This can find many common problems with the configuration.

The most common problems for SSL Orchestrator happened during deployment, when you save the configuration and it is creating the BIG-IP objects, and during operation when you are trying to move traffic.

## Deployment Problems

Deployment problems usually manifest as not being able to save a configuration and seeing various error messages when attempting to do so. Some common causes of these are;

### Addresses, names, or BIG-IP objects already existing on the system

SSLO expects to create a number of objects in BIG-IP, using pre-selected IP addresses and address ranges and using names it chooses. If any of these objects, addresses, or names already exist, it will throw an error and fail to deploy. If this happened, check the BIG-IP configuration for the duplicated object and rename, re-address, or delete it, and try to save the configuration again.

You can also check the logs in `/var/log/restjavad.0.log` and `/var/log/restnoded/restnoded.log` for deployment errors.

### Synchronization issues with HA pairs

The synchronization of SSL Orchestrator uses two different technologies. It uses both the underlying BIG-IP synchronization for syncing BIG-IP objects, and the iAppLX synchronization process for the SSL Orchestrator configuration and objects.

If you suspect sync errors, check the configuration of the BIG-IPs in the sync group to make sure they all match.

You can see synchronization errors in the logs `/var/log/restjavad.0.log` and `/var/log/restnoded/restnoded.log` for synchronization errors.

## Operational Problems

Operational problems happen when SSL Orchestrator has been deployed successfully, however traffic is not acting as expected.

### Testing with traffic

Since in many cases using a browser with a web page will generate a large number of actual requests, it is often easier to test in a more controlled manner.

You can use tools such as `curl` to generate a single request to send through the SSL Orchestrator which makes it easier to spot problems. Some common `curl` commands are:

```
curl -vk https://www.bing.com
curl -vk --proxy 10.20.0.150:3128 https://www.bing.com
curl -vk --proxy 10.20.0.150:3128 --location https://www.bing.com
```

### Narrow down the cause

The first thing to do in this case is to narrow down the cause to the BIG-IP.

- You should initially create a service chain with no services in it and try the traffic without it passing through any other devices.
- If this does not work, the problem is definitely with the SSL Orchestrator
- If this works, then start to add services back into the chain and trying the request again.
- If you find that after adding a specific service back, traffic starts failing;
  - Check that service device for error messages and configuration mis-matches.
  - Check for errors (see enabling debug logging) on the SSL Orchestrator
  - Check using `tcpdump` or another packet capture tool to see if traffic is leaving the SSL Orchestrator and returning as expected.

### Checking Logs

There are several logs that you can use in troubleshooting.

- `/var/log/ltm`: This contains normal traffic errors and problems with the virtual servers and profiles.
- `/var/log/apm`: This contains errors with policy handling.
- `/var/log/restjavad.0.log`: This contains errors with the iAppLX framework.
- `/var/log/restnoded/restnoded.log`: This contains errors when iAppLX is trying to deploy BIG-IP objects.

When looking at logs, `grep` and `tail` are your friends.

`Grep` will search a log and return only the lines that contain a string.

For example, to look for the string “error” in /var/log/apm, you would use the following:

```
grep error /var/log/apm
```

tail will show you the end of a file, and optionally allow the file to update on the screen as new entries are added.

For example, to show the end of the log file /var/log/ltn, and have new entries show up live when they happen, you could use the following:

```
tail -f /var/log/ltn
```

\*Note that there is MUCH more that can be done with these as well as many more resources for filtering logs, see the man pages for these for more information.

```
man tail  
man grep
```

### Enabling debug logging

With debug logging enabled, SSL Orchestrator will log traffic at each step in the process, including the decisions for each of the policy steps. This will generate a large number of log entries, so this should only be left on during the debug process, and not for normal operation.

Debug logging can be enabled in the GUI at SSL Orchestrator / Log / Settings

### Traffic is not blocked when it should be blocked.

If traffic is passing through the system when it should be being blocked. You should first check to make sure that the traffic is actually going through SSL Orchestrator and not being routed around it. Look at the certificate that the client is receiving. Make sure that it is signed by the CA certificate configured on the SSL Orchestrator (for forward proxy) or the certificates used for the inbound traffic (for reverse proxy). If this is not the case, your clients are finding a different path to the servers.

If traffic is passing through SSL Orchestrator, but is not being blocked by a service device, make sure that it is actually going to the service that should be blocking it. Check for the following:

- Check the policy and chain to make sure that the system is configured to send traffic through that service device.
- Check to make sure the service device is not marked down by a monitor. Check in the Local Traffic / Network Map and check for any red diamonds. Then check the setting on the service to see if it is configured to skip that device when down.
- Check traffic statistics to see if packets are going in and out of the virtual server for the service.
- Check any statistics on the service device to see if it shows traffic coming in and out
- Capture traffic on the correct VLAN addressed to the service device using tcpdump or another packet capture tool.

! Remember, not all traffic types will be sent to all service devices, some services can only handle HTTP or HTTPS traffic (for example the HTTP proxy devices).

If traffic is being sent to the service device and returned from it, you will need to troubleshoot the service device to see why it is not being blocked.



If traffic is not being passed to the service device, then it is most likely a problem with the policy.

- Enable debug logging and check the logs at /var/log/apm to see why that device is being bypassed.

### **No traffic is passing through the system**

If no traffic is passing through the system, check the following:

- Check interface and virtual server traffic statistics to make sure that traffic is actually being received by the BIG-IP
- Check for normal addressing and networking issues.
- Check the settings for SNAT and the gateway settings.
- Using curl, try to access the resource directly from the SSL Orchestrator system.
- If using explicit proxy, check the proxy settings on the clients to make sure they are pointing to the proxy address.
- Capture traffic entering and leaving the BIG-IP to make sure that requests are entering and leaving AND that responses are entering and leaving. (a common mistake is to only check one side).

### **Only Select traffic is passing through the system**

If only certain traffic is passing through the system, you can check the following:

- Check the certificate for the traffic to make sure it is not expired and is valid
- Check to make sure the certificate is from a trusted CA and that CA is in the CA bundle being used.
- Check to make sure the certificate is not being invalidated by CRL or OCSP
- Check to make sure the traffic is not being blocked by a service device (this can be done by removing the service devices from the chain one by one).
- Check and clear the certificate cache and try again using:

```
tmssh show ltm clientssl-proxy cached-certs clientssl-profile [CLIENTSSL PROFILE]
virtual [INGRESS TCP VIP]
```

```
tmssh delete ltm clientssl-proxy cached-certs clientssl-profile [CLIENTSSL PROFILE]
virtual [INGRESS TCP VIP]
```

- Check for SSL/TLS errors in /var/log/ltm after changing the ssl log level.
- Check the traffic to see if it uses a pinned certificate. (see Certificate Pinning and HPKP)
- Using a service like <https://www.ssllabs.com/ssltest/> check the sites ssl requirements and certificate.
- Check any URL categories used to see if the URL matches.

## Contacting F5 Technical Support

Prior to contacting F5 technical support, please make sure you have read the Instructions here:

<https://support.f5.com/csp/article/K2633>

In addition to the information in that article:

- Make sure your system is backed up.
- Make sure you have documented any modifications to your system.
- If you have disabled strict update and made modifications, inform technical support of these.
- If these were based on this document, also inform technical support of that fact.
- F5 support may ask you to re-enable strict update, which will overwrite any modifications made outside of the SSL Orchestrator interface.

## Use Cases

This section provides examples of various use cases for SSL Orchestrator v4.0. These are intended to be examples of best practices, but in many cases will not directly work without customization for the specific deployment need.

### Supportability of Changes and Flagging Changes

Making changes to any objects that are marked as “strict update” will normally not be supported, and F5 support may ask that you re-enable strict update before troubleshooting SSLO, this will overwrite the changes and reset SSLO to the configuration deployed by the factory.

The exception to this is when using specific workarounds to issues that F5 has published and marked as “official”. (including the ones in this document) If you implement these AS DOCUMENTED, they will be considered supported. In order to identify these on a system, the description field of the object that you disabled strict update on and are modifying should have a string such as “MOD:<modification name>” added to it. For example, if you added SNAT to a service, in the service description, you would add “MOD:ADDED-SNAT”. This will allow F5 support engineers (and you) to identify which objects have been modified using official workarounds.

### Using the SSLO Extras Library

Many of the use cases listed use the SSLO Extras Library. This is a library of additional helper iRules that can be added to the system to in common use cases.

To use this library,

1. Navigate to Local Traffic / iRules and create a new iRule.
2. Name the iRule “sslo\_extras”.
3. Copy and paste the rule from the appendix into the Definition field.
4. Save the iRule.

If you need to use a different name for this, or save it in a different partition, you will need to modify the use case example to match the new name and/or partition.

## Standard Use Cases

### Block HTTP and HTTPS traffic based on URL

The best approach to blocking traffic due to a URL is to use URL categories. If URL Filter or SWG is licensed and setup, you can make decisions based on pre-defined categories, however, even if these are not available, you can create custom categories with URL's you define and use those in decision making.

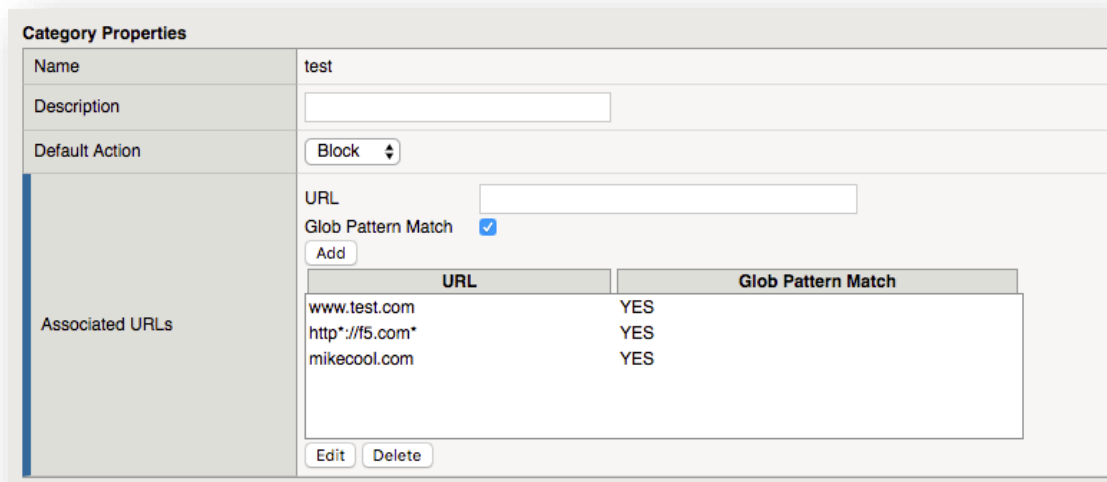
! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

The URI that is matched is based on the "Category Lookup" agent option set in the Categorization Macro, (see the SSL Orchestrator 4.0 Reference Guide for more information on the selection options and how they relate to what is matched.)

#### Creating a custom category

! This is only needed if you are not filtering based on URL Filtering or SWG Categories.

1. Navigate to SSL Orchestrator / Policies / URL Categories
2. Click on [Create]
3. Enter the name and the list of URLs to block (see BIG-IP Secure Web Gateway Documentation for more information on this step)



**Category Properties**

Name	test								
Description									
Default Action	Block								
Associated URLs	<div>URL <input type="text"/></div> <div>Glob Pattern Match <input checked="" type="checkbox"/></div> <div>Add</div> <table border="1"><thead><tr><th>URL</th><th>Glob Pattern Match</th></tr></thead><tbody><tr><td>www.test.com</td><td>YES</td></tr><tr><td>http*://f5.com*</td><td>YES</td></tr><tr><td>mikecool.com</td><td>YES</td></tr></tbody></table> <div>Edit Delete</div>	URL	Glob Pattern Match	www.test.com	YES	http*://f5.com*	YES	mikecool.com	YES
URL	Glob Pattern Match								
www.test.com	YES								
http*://f5.com*	YES								
mikecool.com	YES								

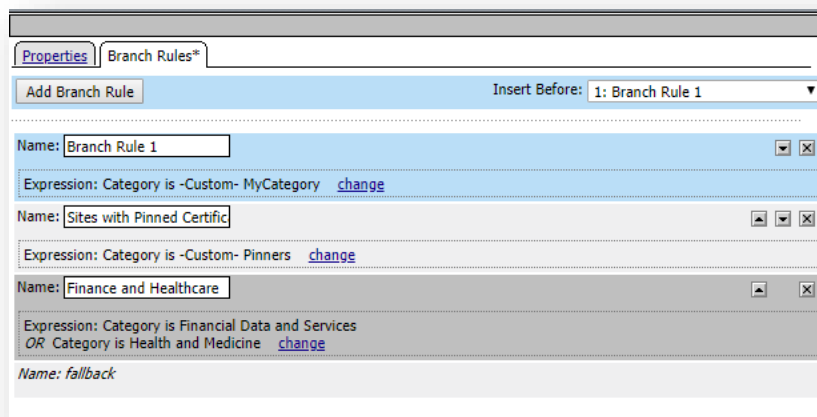
4. Save the new category.

## Setting up the policy

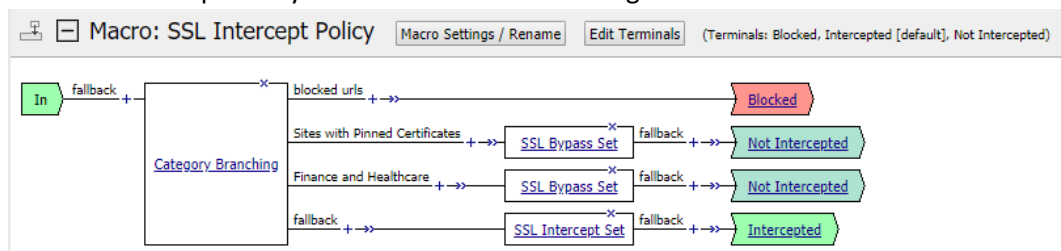
Once you have the categories setup you need, you will need to setup the policy.

! If you wish to also categorize HTTP (non-encrypted) traffic, you will need to enable HTTP categorization (See Categorize URLs for HTTP (non-encrypted) traffic)

1. Navigate to SSL Orchestrator / Policies / Per-Request Policies
2. Find the Per-Request Policy that you need to modify and show all APM Per request policies using the "+ Show All" button.
3. Select the "ssloP\_<chain name>\_prpTcp" APM Per Request Policy.
4. Expand the "SSL Intercept Policy" macro.
5. Edit the "Category Branching" agent and select the "Branch Rules" tab.
6. Click on [Add Branch Rule]
7. Provide a descriptive name.
8. Click on the "change" link next to "Expression: Empty"
9. Click on [Add Expression]
10. Set the Agent Sel: and Condition to "Category Lookup"
11. Select the category of URLs that you wish to block and save
12. Your agent should now look something like:

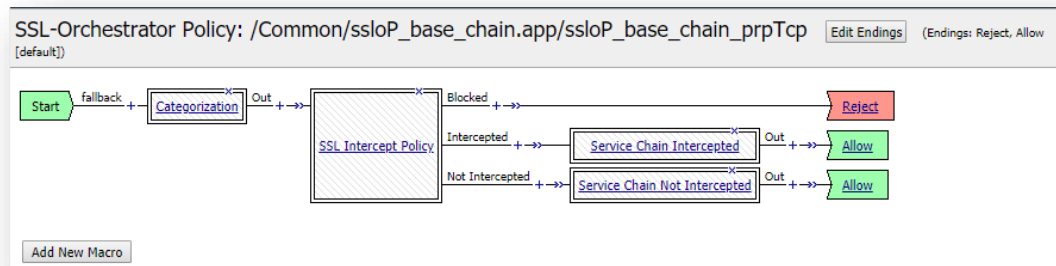


13. Add a "Blocked" terminal to the macro by clicking on the "Edit Terminals" button.
14. Modify the terminal of the new branch you just created to "Blocked" by clicking on the existing terminal
15. Your SSL Intercept Policy Should now look something like:



16. Scroll to the top of the policy and find the base policy,
17. Change the ending of the "Blocked" terminal to "Reject"

18. Your base policy should now look like this:



## Select a different intercept chain based on IP intelligence or geolocation

In this example, we want to select a different chain that provides additional scanning for traffic from North Korea or traffic that has a suspect reputation.

In order to use IP Intelligence information, you will need to have IP Intelligence licensed and configured (See the BIG-IP documentation for more information).

In order to use IP Geolocation, you will have to have it configured. (See BIG-IP documentation for more information).

The “IP Policy” macro by default is aimed at blocking specific traffic, but we will modify this to instead set a flag indicating that we wish to use a different chain. Then we will build the new chain macro and link it to the SSL Intercept policy by looking for the variable we set.

! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

### Open the Policy

1. Before starting, make sure that you have the required services setup for the enhanced scanning.
2. Navigate to SSL Orchestrator / Policies / Per-Request Policies
3. Find the Per-Request Policy that you need to modify and show all APM Per request policies using the “+ Show All” button.
4. Select the “ssloP\_<chain name>\_prpTcp” APM Per Request Policy.

### Modifying the IP Policy Macro

1. Expand the “IP Policy” Macro
2. Click on the “Macro Settings / Rename” button
3. Change the “One-Shot” macro option to “Yes”
4. Click on the “Server IP Geolocation Match” agent and the “Branch Rules” tab
5. Change the name of the branch to “IP Geolocation Country code is KP”
6. Click on the “Change” link for that branch and modify the Country code to “KP” (the ISO 3166-1 alpha-2 country code for North Korea)

7. Save the expression and click “finish” to save the agent.
8. Click on the “Server IP Reputation” agent and the “Branch Rules” tab
9. Change the branch name “Good” to “Questionable” (This branch by default assumes that if there is no information for address, it’s a good one. We want to assume that it’s questionable and do enhanced scanning)
10. Save the agent
11. Change the “Failed” terminal to “Passed” (You might want to keep a “Failed” option for certain things, but in this example, we are not blocking anything, simply doing more scanning)
12. On the fallback branch from the Server IP Reputation agent, add a new “Variable Assign” agent by clicking on the “+” link.
13. Change the name to “variable-normal scanning”
14. add a new variable entry and click “change” to edit it.
15. On the left, In the custom variable field, enter “session.custom.scan\_level”
16. On the right side, change the type to “Text”, and enter “normal”
17. your agent should now look like this:

The screenshot shows a configuration window for a 'Variable Assign' agent. The 'Name' field is set to 'variable - normal scanning'. Under the 'Variable Assign' section, there is an 'Add new entry' button and an 'Insert Before' dropdown set to '1'. Below this is a table with one entry:

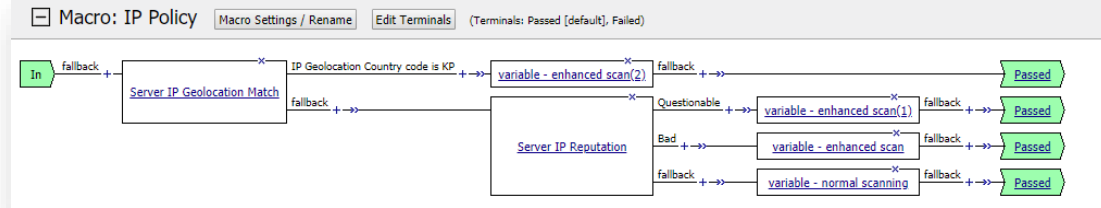
	Assignment	
1	session.custom.scan_level = Text Normal <a href="#">change</a>	<a href="#">X</a>

18. Save the agent
19. On all three of the other terminal links, add another “variable assign” agent and set the name to “Variable – enhanced scan, and the assigned text to “Enhanced” These should each look like:

The screenshot shows a configuration window for a 'Variable Assign' agent. The 'Name' field is set to 'variable - enhanced scan'. Under the 'Variable Assign' section, there is an 'Add new entry' button and an 'Insert Before' dropdown set to '1'. Below this is a table with one entry:

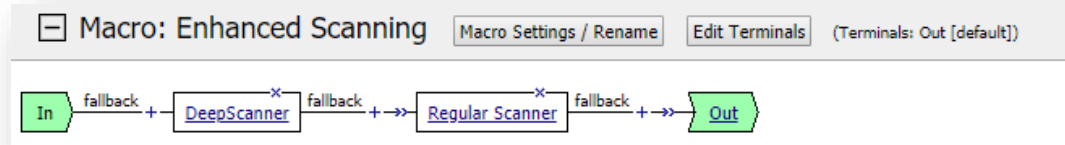
	Assignment	
1	session.custom.scan_level = Text Enhanced <a href="#">change</a>	<a href="#">X</a>

20. Your policy should now look like:



### Building the new Intercept chain macro

1. On the core policy, the [Add New Macro] button to add a new macro.
2. Name the macro “Enhanced Scanning Intercepted Chain” and base it on the “Empty” template
3. Save the new macro
4. Find the new “Enhanced select Scanning Intercepted Chain” macro and expand it.
5. For each service in the intended chain, do the following. If order matters, you need to add the services in the order in which you wish them run.
  - a. Add a new service (in the order in which you wish them run)
    - i. Click on the “+” after the prior service
    - ii. Add a Service Connect type agent (from under the Traffic Management Tab)
  - b. Select the connector object for the first service in the chain.
    - i. Most the connector objects are named as follows:  
 “/Common/ssloS\_<service\_name>-[t/u]-connector” with the “t” or “u” indicating TCP or UDP support.
  - c. Name the connector after the service and save it.
  - d. Repeat until all services have been added.
6. Your new macro should look like this:



7.

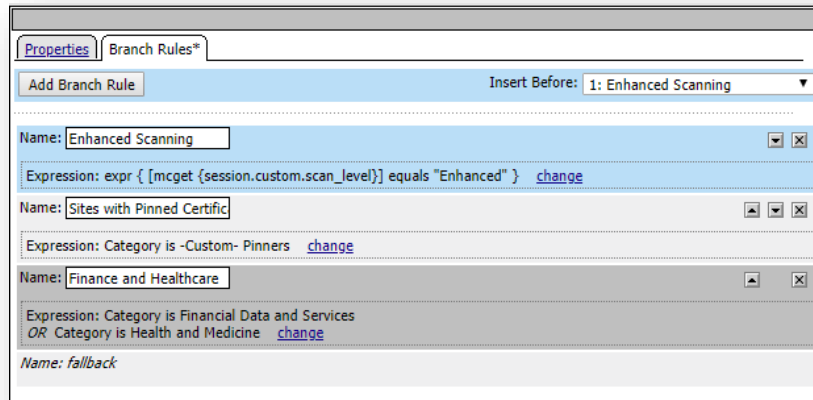
### Setting up branching for the new intercept macro

1. Find and expand the SSL Intercept Policy Macro
2. Click on the Category Branching Agent and “Branch Rules” tab
3. Add a new branch rule and call it “Enhanced Scanning”
4. Edit the expression using the “edit” link and change to the “Advanced” tab
5. Add the following expression to the text box:

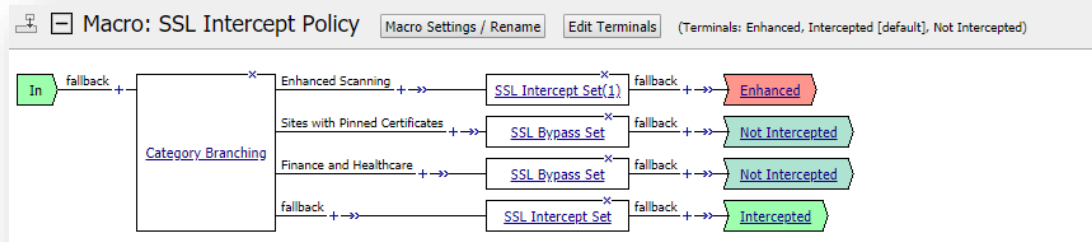
```
expr { [mcget {session.custom.scan_level}] equals "Enhanced" }
```



6. Your agent branching rules should now look like this:



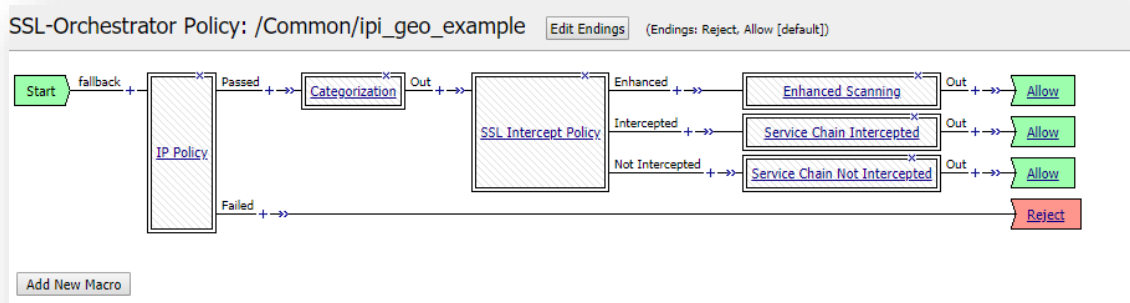
7. Save the agent config.
8. On the “Enhanced Scanning” branch, add a “SSL Intercept Set” agent using the “+” link.
9. Save the new agent
10. Using the “Edit Terminals” button, add a new terminal called “Enhanced”
11. Set the terminal on the “Enhanced Scanning” branch to “Enhanced”
12. The SSL Intercept Policy macro should now look like:



### Tie it all together in the base policy

1. Scroll up to the base policy
2. Add the IP Policy object (from the SSLO macros tab) in before the “Categorization” macro using the “+” link.
3. Add the “Enhanced Scanning Service Chain” macro (from the SSLO Macros tab) on the “Enhanced” line after the SSL Intercept Policy.
4. While not needed, let's fix the “Failed” terminal from the IP Policy so that it rejects traffic.
  - a. Click on the “Allow” ending on the “Failed” branch from the “IP Policy” macro and change it to “Reject”.
  - b. Since we did not use the “Failed” branch, this should have no impact, but if in the future it is used, this will make sure it is correctly handled.

5. Your base policy should now look like:



## Select the Non-Intercept chain based on IP information

In this example, we wish to have the systems that are in a specific IP network use the non-intercept chain, so their traffic is not decrypted. This might be the case if we had some payment processing servers that we still wanted to scan for attacks but did not want to decrypt.

We will build a new macro for the IP address filtering, we could use the IP Services macro and replace the items in it if we wished, but we will show how to build one from scratch.

For performance reasons, it's important that IP branching be done in its own macro, it does not need to be done in the "IP Services macro", however, the macro needs to have the "One-Shot" macro option set to "Yes". This option allows that macro to only be run once per TCP connection, instead of running it every request, saving a significant amount of time.

Also, in this example, we will bypass the categorization steps for any matching traffic since we already know what type of traffic it is. In a live configuration, this might not be the case.

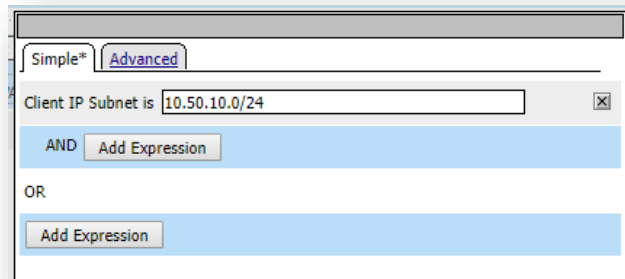
**!** Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

## Open the Policy

1. Before starting, make sure that you have the required services setup for the enhanced scanning.
2. Navigate to SSL Orchestrator / Policies / Per-Request Policies
3. Find the Per-Request Policy that you need to modify and show all APM Per request policies using the "+ Show All" button.
4. Select the "ssloP\_<chain name>\_prpTcp" APM Per Request Policy.

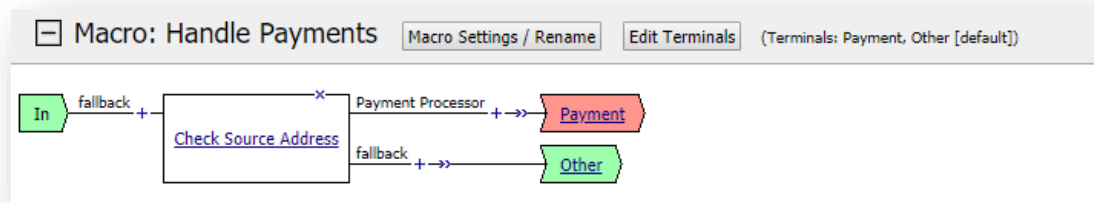
## Create a Handle Payments Macro

1. Create a new macro by clicking on the "Add New Macro" in the base policy
2. Use the "Empty" macro template and name this "Handle Payments"
3. Find and expand the "Handle Payments" macro
4. Add an "Empty" macro and name it "Check Source Address"
5. Click on the "Branch Rules" tab and add a new branch
6. Call this branch "Payment Processor" and add an expression
7. In Agent Sel:, Select "IP Subnet Match" (we could pick "Port match" if we wanted to match a TCP or UDP port)
8. In condition, select "Client IP Subnet Match" (since we want to detect traffic FROM the payment processors)
9. In the "Client IP Subnet is", add the network that contains the processors. (We will use 10.50.10.0/24, but in a live system this should either be a single system with a /32 mas , if IPv4, or a smaller network range)
10. Your branch rule should now look like:



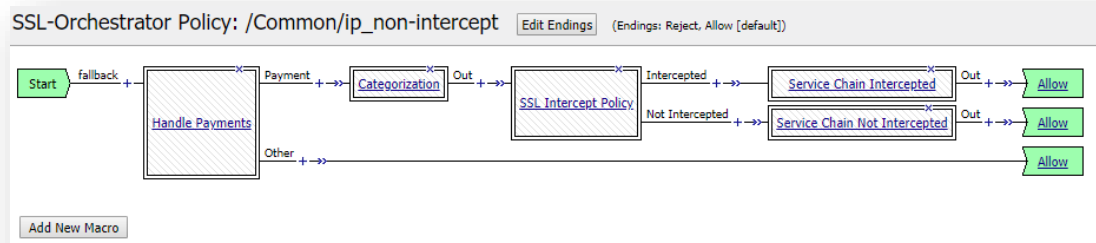
- 11.
12. Save the agent
13. Edit the terminals using the "Edit Terminals" button
14. Rename the "Out" terminal to "Other", and add a "Payment" terminal
15. On the "Payment Processor" branch, change the terminal to "Payment"

16. Your “Handle Payments” macro should now look like this:

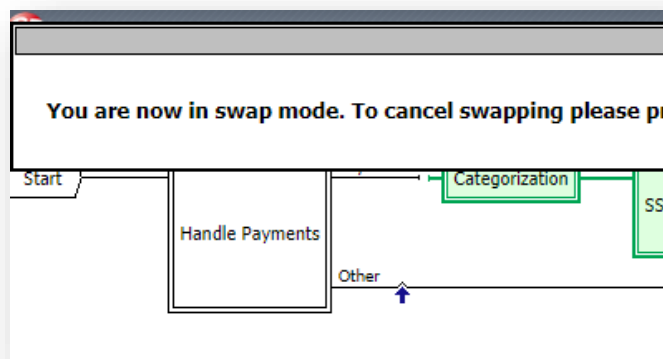


### Integrate into the base policy

1. In the base policy, add the Handle Payments Macro before the Categorization macro using the “+” link.
2. At this point, your base policy may look like this:

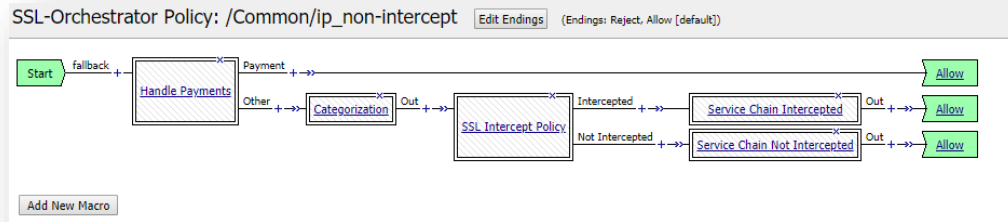


3. If the “Payment” terminal goes to the “Customization” macro, that isn’t right, we want the “Other” terminal going to “Customization”. If this is the case for you:
  - a. Click on the swap link (the “>>”) next to the Categorization macro



- b. Now click on the arrow next to the “Other”

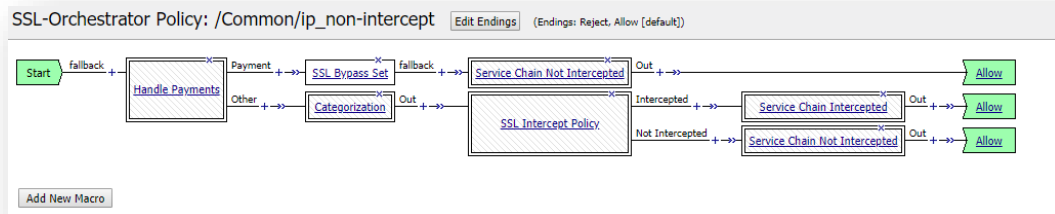
c. You should now see something like this:



Now, let's add in the non-intercept chain again for the payment branch.

Before passing any traffic to a chain, intercept or non-intercept, you always need to pass it through a “SSLO Intercept Set” or “SSLO Bypass Set” agent.

4. Add a new agent on the “Payment” branch by clicking the “+” and selecting the “SSLO Bypass Set” agent.
5. Save the agent
6. Add the “Service Chain Not Intercepted” macro after the bypass set you just added using the “+” link.
7. Your base policy should now look like this:



Remember, since we bypassed the categorization macro for this, we would be unable to do any categorization activities based on URL or protocols.

## HTTP should bypass all chains based on a user's group

In this example, we would like any users that are a member of the “no\_intercept” group to bypass SSLO completely.

This example assumes that the system has APM licensed and is setup for authentication (See Setting up authentication with explicit proxy connections).

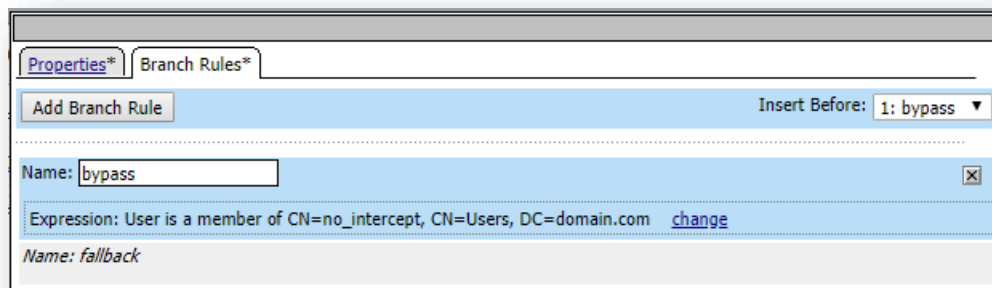
Since we are bypassing everything, we will create a branch before the categorization step for this.

! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

1. Before starting, make sure that you have the required services setup for the enhanced scanning.
2. Navigate to SSL Orchestrator / Policies / Per-Request Policies
3. Find the Per-Request Policy that you need to modify and show all APM Per request policies using the “+ Show All” button. Select the “ssloP\_<chain name>\_prpTcp” APM Per Request Policy.
4. Add an “Empty” agent just prior to the “Categorization” macro using the “+” link.
5. Name the agent “Check Bypass Group” and select the “Branch Rules” tab.
6. Add a new Branch Rule and name it “bypass”.
7. Edit the expression for the rule using the “change” link.
8. Select the advanced tab and enter the following in the text box:

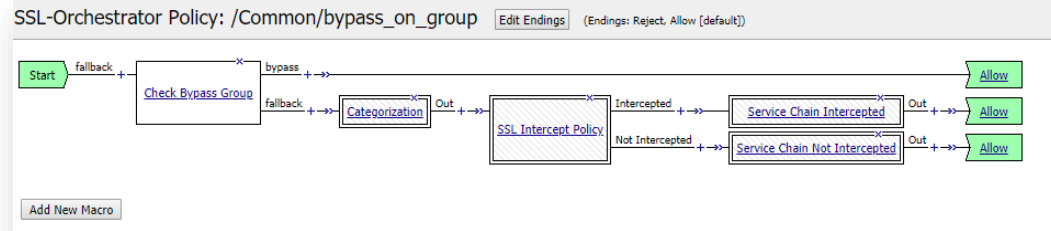
```
expr { [mcget {session.ad.last.attr.memberOf}] contains "CN=no_intercept, CN=Users, DC=domain.com" }
```

9. Your agent should now look like:



10. Save the agent.

11. Your base policy should now look like:



## Remap ports for inbound traffic

In order to use the “Port Remap” option for inbound traffic, you need to add an iRule to the interception rule to allow the system to correctly do this.

### Updating the interception rule

! Note, this assumes that you selected “Advanced” configuration when creating the inbound rule, if you did not, you will not see the iRules and will have to recreate the interception rule.

1. Install the SSLO Extras library (see Using the SSLO Extras Library)
2. Navigate to Local Traffic / iRules
3. Create a new iRule called “allow\_remap\_ports” and add the following to it.

```
When CLIENT_ACCEPTED { call /Common/sslo_extras/set_ctx_to_https }
```

4. Navigate to SSL Orchestrator / Deployment / Interception Rules
5. Select the inbound interception rule that contains the service in question.
6. Find the Resources / iRules section, and add the iRule to the interception rule.

## Categorize URLs for HTTP (non-encrypted) traffic

ID 740433

By default, HTTP Traffic that is not encrypted (so, non-HTTPS) will not be categorized by the SSL Orchestrator.

Category Lookup agent works with SNI and HTTP Connect Hostname but cannot match the URL database against an HTTP Host header.

To work around this, we can mark the category during the initial TCP connection. However, If the URL changes mid-connection this workaround will not catch that. This should be an unusual case and would require the following to all be true:

1. There must be more than one host using the same IP address.
2. Both hosts would categorize differently within URL categorization and you need to differentiate between them.

3. Both are using HTTP (not HTTPS)
4. An agent goes to one immediately after the other (without requesting a different host first)
5. The agent uses the same TCP connection for both.

While item 1 is common, items 2-5 are not common, and item 6 is extremely uncommon and may be an indication of some kind of negative activity.

- ! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects
- ! For this to work with non-custom categories, URL Filtering or SWG must be licensed and configured.
- ! If you add a new custom URL category, or if the URL category list changes, you will need to update the data group.

#### How it works:

Before any traffic is processed, we need to create a data group of the URL categories, then For each request the system will

1. Update a variable with the hostname from the HTTP URI (in the “store\_host\_header” iRule)
2. Branch HTTP traffic off in the APM Per-Request policy, Categorization Macro
3. Run the “get\_urlcat” iRule on the traffic, matching the URI to the data group and updating an APM variable.
4. Copy that variable into the APM variable that the URL branching information uses in the next macro.

#### Create Data Group:

1. Access the CLI and login with root account
2. Create a data group of URL categories based on the configuration of the system. (note, if you plan on using non-custom categories, make sure that SWG or URL Filtering is licensed and configured, and that the categories have been downloaded. (See the BIG-IP documentation for more information)
3. The following script will create the data group the first time.

```
tmsl list sys url-db url-category |grep -E 'sys url-db url-category|
cat-number' |sed -e 's/sys url-db url-category //g;s/    cat-number //g;s/
{ //g' |sed "N;s/\n/ := /" |sed -e 's/$/,/' > /var/tmp/urlcat.txt

tmsl create /sys file data-group urlcat_file separator "==" source-path
file:/var/tmp/urlcat.txt type string

tmsl create /ltm data-group external urlcat_dg external-file-name
urlcat_file
```



- And this one can be used to update the datagroup when the URL categories change.

```
tmslsh List sys url-db url-category |grep -E 'sys url-db url-category|
cat-number' |sed -e 's/sys url-db url-category //g;s/ cat-number //g;s/
{ //g' |sed "N;s/\n/ := /" |sed -e 's/$/,/' > /var/tmp/urlcat.txt

tmslsh modify /sys file data-group urlcat_file separator "==" source-path
file:/var/tmp/urlcat.txt

tmslsh modify /ltm data-group external urlcat_dg
```

### Copy and update the system iRules

1. Navigate to Local Traffic / iRules
2. Install the SSLO Extras Library (see Using the SSLO Extras Library)
3. Create an iRule and name it "sslo\_get\_category"
4. In the "definition" field, add the following text:

```
when ACCESS_PER_REQUEST_AGENT_EVENT { call /Common/sslo_extras/get_urlcat }
```

5. Save the iRule.
6. Open the iRule named "sslo\_<system name>-in\_t"
7. Copy the contents from the iRule
8. Close the iRule and create a new iRule named <sslo\_system name>-in\_t-copy
9. Paste the contents from the system iRule in the new iRule.
10. Find the section "when CLIENT\_DATA"
11. Within that section, immediately after the "set said [TCP::payload] (and before the closing "}") add:

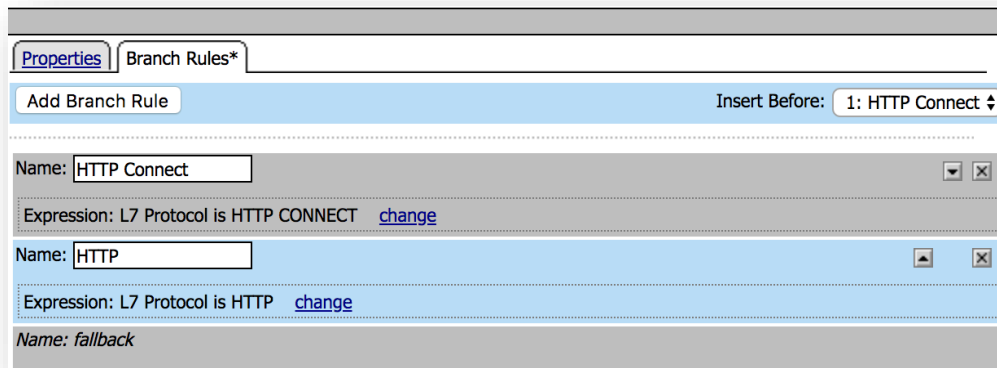
```
## MOD: Call update host headers for http categorization
call /Common/sslo_extras/store_host_header $said
## END MOD
```

12. Save the iRule

### Update the APM Per-Request Policy

1. Navigate to SSL Orchestrator / Policies / Per-Request Policies
2. Find the Per-Request Policy that you need to modify and show all APM Per request policies using the "+ Show All" button.
3. Select the "ssloP\_<chain name>\_prpTcp" APM Per Request Policy.
4. Find and expand the Categorization macro
5. Open the "L7 Protocol Lookup" agent and select the "Branch Rules" tab.
6. Add a branch to the L7 Protocol Lookup and name it "HTTP"
7. Edit the expression using the "change" link
8. Set "Agent Sel" to "L7 Protocol Lookup"
9. Set "Condition" to "L7 Protocol"
10. Set "L7 Protocol Is" to "HTTP"
11. Click Add Expression
12. Move this branch below the "HTTP CONNECT" one using the down arrow.

13. Your agent should now look like:



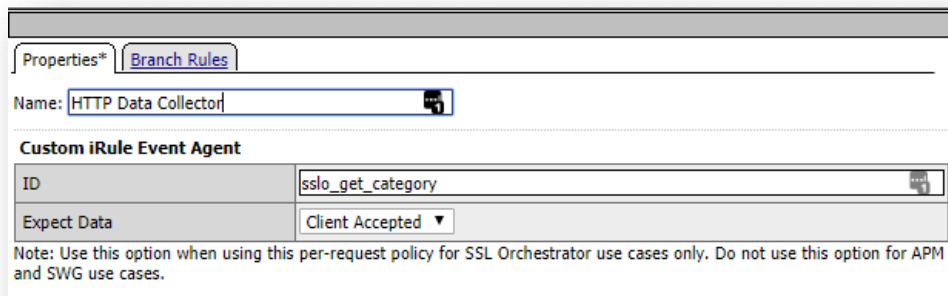
14. Save the agent.

15. Create an iRule Event agent on the "HTTP" branch using the "+" link.

16. Name this event "HTTP Data Collector"

17. Set the ID to "sslo\_get\_category" and the "Expect Data" to "Client Accepted"

18. Your agent should look like this:



19. Create a "Variable Assignment" agent on the same branch after the "HTTP Data Collector" agent.

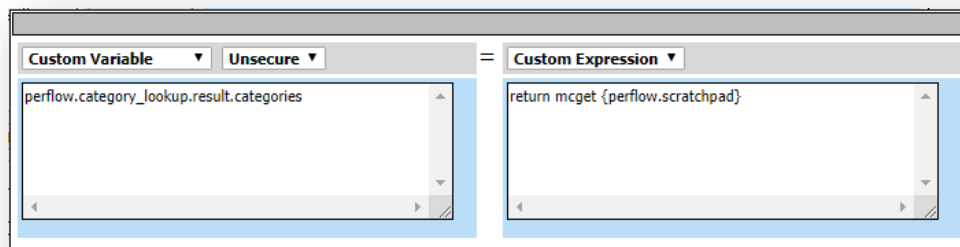
20. Name this agent "HTTP Host Variable Assign" and add a new entry

21. Edit the entry using the "change" link.

22. In the left box, add "perflow.category\_lookup.result.categories"

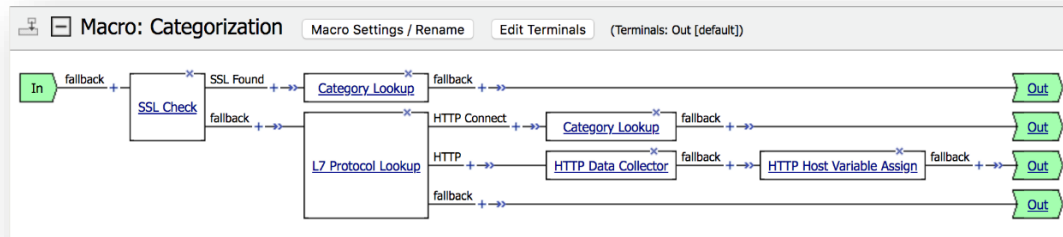
23. In the right box, add "return mcget {perflow.scratchpad}"

24. Your assignment should now look like this:



25. Save this and save the agent

26. Your Categorization Macro should now look like:



## Setup without URL Filtering or SWG licensed

ID 740435

By default, the system assumes that URL Filtering or SWG is licensed and provisioned. However, if URL Filtering or SWG is not provisioned or the license is expired, then the policy will not forward any traffic. To use SSL Orchestrator without licensing URL Filtering or SWG, follow the instructions below.

! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

### Configuration

1. Navigate to SSL Orchestrator / Policies / Per-Request Policies
2. Find the Per-Request Policy that you need to modify and show all APM Per request policies using the "+ Show All" button.
3. Select the "ssloP\_<chain name>\_prpTcp" APM Per Request Policy.
4. In the Categorization Macro, open both of the "Category Lookup" Actions.
5. Verify that they were changed to "custom categories only" (this should have happened when opening them).
6. Save the Action.
7. Close the Policy

# Configuring Inbound Services when SNAT is used

ID 740439

If SNAT is enabled, it is currently configured globally for both inbound and outbound traffic, however when both inbound and outbound traffic are configured, the SNAT addresses are only on the outbound side of the system. The inside hosts should not have a route to these addresses directly, causing traffic to not be able to return.

- ! If SSL Orchestrator is setup to ONLY handle inbound OR outbound traffic, this is not an issue.
- ! If SNAT is not configured, this not an issue.

The solution to this is to manually create a SNAT pool and enable SNAT on inbound virtual servers.

- ! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

## Setup the SNAT Pool

1. Navigate to Local Traffic / Address Translation / SNAT Pool List
2. Create a SNAT pool named "sslo\_inbound\_snat"
3. Add the SNAT addresses you wish to use to the pool
4. Save the pool by clicking "Finished"

## Unlocking the application

1. Navigate to SSL Orchestrator / Deployment / Deployment Settings
2. Disable "Strict Update"
3. Add the following to the Description field: "MOD:INBOUND\_SNAT"

- ! This tells F5 support that you are using the official modification for this workaround.

4. Save the application using the "finished" button.

## Apply the rules to the virtual servers

5. Navigate to Local Traffic / Virtual Servers
6. Open the virtual server which has the same name as the inbound rule
7. Enable SNAT on this virtual server and select the SNAT pool you created.
8. If needed, repeat for other inbound virtual servers.

## Configuring Layer 3 and HTTPS services for inbound traffic

---

ID 740442

When configuring inbound traffic, and Layer 3 or HTTPS services are used, you may need to take some additional steps to configure this combination.

Regardless of traffic flow, packets always enter an L3 service on its “inbound” side, so for a simple Layer 3 or HTTP service that has a single default route back to the F5, will return to the incorrect interface.

- ! If SSL Orchestrator is setup to ONLY handle inbound OR outbound traffic, this is not an issue.
- ! This is only an issue for HTTPS or Layer 3 services.

### Configuration Options

There are two options for this:

1. Setup the service device to route traffic back based on the destination address. This would likely require setting up something like “policy-based routing” on the service device. You should contact the service device vendor for assistance on this.

-OR-

2. Create multiple services for the security devices but swap the to/from service Networks on them.

## Adding an iRule to a service

---

ID 742607

In most of the services, there is a field under the “Resources” section to add an iRule to the service. When you add this rule however, it is not saved with the service.

Make sure that any iRules that you add do not modify the traffic flow (such as changing the node, pool, SNAT, or virtual servers) as these types of actions will break the service operation. These iRules are intended to handle application layer protocol manipulation only, such as inspecting and modifying HTTP headers or payloads.

- ! Note: In many cases it may be easier to simply add the rule to the policy as an iRule agent or to one of the interception rules
- ! Review and be aware of the known issues when editing an SSLO deployed object. (see Modification of SSLO deployed objects)

### Steps

1. Navigate to SSL Orchestrator / Services
2. Select the service type and service to modify
3. Disable “Strict Update”
4. Add the following to the end of the comment field “MOD:CUSTOM\_IRULE”

! This allows you and F5 support to know why strict update was disabled.

5. Click Finished on the service.
6. Note the name of the service.
7. Navigate to Local Traffic / Virtual Servers
8. Find the virtual server whose name matches “ssloS\_<service name>\_t” **AND/OR** ssloS\_<service name>\_u” (for UDP traffic, if needed) and open it.
9. Add the iRule to the virtual server / resources tab.

## Adding a monitor to a TAP service

ID 742608

By default, TAP services do not define monitors since they are not inline services. Therefore, the service device going down would not cause traffic to fail. However, in some cases, you may wish to block traffic if this service has failed.

! In the TAP service there is an option for "action on service down", however, since there is no monitor attached to the tap service, anything selected here will be ignored.

### What this does

In this use case, we will add an ICMP monitor and use an alias to define what address to ping.

Since most monitoring devices using TAP services will not have an address assigned to the interface that they are monitoring on, you will have to use a different address for the alias. This might be the management address, or it could be another address on the system.

! Since the traffic may be taking a different route to get to the monitored device, you are NOT monitoring the network or the receiving device, you are only monitoring access to the interface you are pinging. This could miss critical outages if they involve the network path between SSLO and the monitoring interface.

! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

### Create the Monitor

1. Navigate to Local Traffic / Monitors
2. Add a new monitor and name it based on the service you will be attaching it to.
3. Select the “Gateway ICMP” monitor type
4. Set field “Transparent” to “Yes”
5. Add the service device’s address to the “Alias Address” field.
6. Save the monitor

### Update the Service

1. Navigate to SSL Orchestrator / Services / TAP Services
2. Select the service you wish to add the monitor to

3. Disable “Strict Update”
4. Add the following to the end of the comment field: “MOD:ADD\_MONITOR”

! This allows you and F5 support to know why strict update was disabled.

5. Set the service down action as needed
6. Save the service

### Assign the monitor to the Virtual Server

4. Navigate to Local Traffic / Pools
5. Find the pool whose name matches the service and open it.
6. Add the health monitor you created

## Configuring SSLO to use an upstream explicit proxy

---

To configure explicit proxy chaining (sending data to an upstream explicit proxy device), you need to add the Proxy chaining macros into the default policy.

To correctly send traffic to an upstream explicit proxy, we need to add the HTTP Connect header to the traffic, as well as not re-encrypt it before sending it.

This is done by adding two of the unused macros in the default policy into the policy. The Proxy Chaining(Connect) macro will initiate the connection with the upstream proxy to verify and forge the end certificate and the Proxy Chaining(URI Rewrite) macro will actually rewrite the headers for the connection.

! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

### Creating the Pool

1. Navigate to Local Traffic / Pools.
2. Create a new pool named “sslo\_xp\_pool”.
3. Add a “gateway\_icmp” health monitor.
4. Add the upstream explicit proxy addresses to the members list.
5. Save the pool

### Modifying the Policy

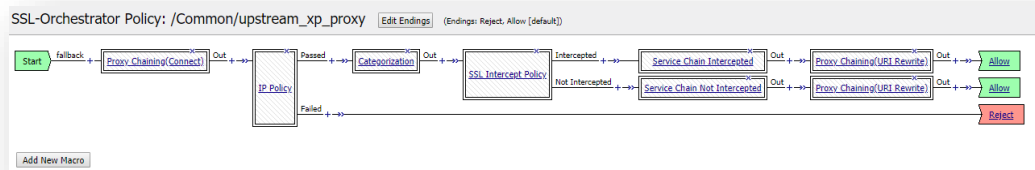
1. Navigate to SSL Orchestrator / Policies / Per-Request Policies
2. Find the Per-Request Policy that you need to modify and show all APM Per request policies using the “+ Show All” button.
3. Select the “ssloP\_<chain name>\_prpTcp” APM Per Request Policy.
4. Find and expand the Proxy Chaining(Connect) Macro
5. Open the “Proxy Connect” agent.
6. Set the pool to “/Common/sslo\_xp\_pool”
7. Change the upstream Proxy Mode to Explicit(No URI Rewrite)

! If this step is missed, only HTTPS traffic will actually pass through the proxy.

8. Save the agent.
9. Find and expand the Proxy Chaining(URL Rewrite) Macro
10. Open the “Proxy URI Rewrite” agent.
11. Set the pool to “/Common/sslo\_xp\_pool”

! You do not need to add any custom headers to the configuration, SSL Orchestrator will automatically handle this.

12. Scroll up to the base policy
13. Add the Proxy Chaining(Connect) Macro before the Categorization macro using the “+” link
14. On all branches that allow traffic (That use the “Allow” ending):
  - a. Add the Proxy Chaining(URL Rewrite) Macro after the final macro or agent before the “Allow” using the “+” link
  - b. Repeat for each “Allow” path.
15. Your policy should look like this:  
(in this example, we also have the IP Policy to show a “Reject” link, but the IP Policy macro is not required for this use case)



## Setting up authentication with explicit proxy connections

This use case provides for users to authenticate to SSL Orchestrator before passing through the services. The authentication information can then be used in policy branching decisions.

- ! Note that you must have APM licensed and configured in order to support authentication.
- ! SSL Orchestrator only supports authentication on explicit proxy connections, not transparent proxy ones.

To setup authentication of users with explicit proxy connections. You will need to create a SWG-Explicit profile. Note that the below instructions are for a basic profile and policy using the local authentication database, you will probably need to modify these to fit the specific use case. (See APM documentation for more information)

- ! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects



## Create local authentication database

*This step is not needed if you are configuring an external authentication system.*

1. Navigate to Access / Authentication / Local User DB / Instances
2. Create a new local authentication instance named "localdb"
3. Navigate to Access / Authentication / Local User DB / Local DB Users
4. Add users as needed to the system
  1. Click on "Create New User"
  2. Provide a username and password
  3. Make sure that the instance you just created is selected ("localdb").
  4. Click "OK" to save.
  5. Repeat for additional users.

## Configure APM Authentication policy

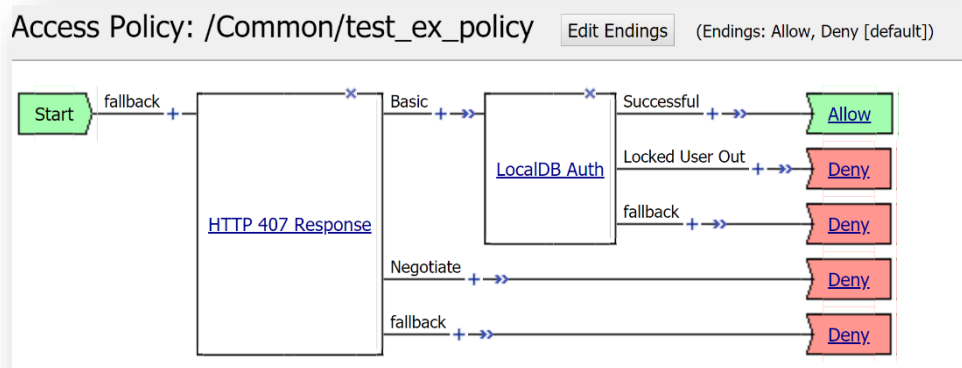
! This example is both insecure and uses a local database for auth, we recommend that you use more secure methods when creating your policy. See the APM documentation for more information.

1. Navigate to Access / Profiles/Policies / Access Profiles.
2. Click Create to create a new profile.
3. Under Profile Type, select SWG-Explicit
4. Name the profile "sslo\_xp\_session"
5. Add the correct language under Language Settings / Languages.
6. Save this profile by clicking on "finished"
7. Open the per session policy in the Visual Policy Editor by selecting "edit" in the per-session policy column.
8. Add a "HTTP 407 Response" agent by clicking on the "+" after the start action and selecting HTTP 407 Response.
9. In the "Basic Auth Realm" field, enter "SSLO"
10. In the "HTTP Auth Level", select "basic"
11. Click on the "Branch Rules" tab.
12. Remove the "Negotiate" branch

! This will lock the authentication into Basic auth, which is an insecure authentication mechanism. F5 strongly recommends not using this in production systems.

13. Save the action.
14. Add an Authentication action on the "basic" branch by clicking on the "+" next to "Basic" and selecting the Authentication tab, and "LocalDB Auth"
15. Select the Local DB instance that you just created "localdb" and save the action.
16. Modify the ending for "Successful" auth to "Allow"

Your policy should now look like:



### Update the Interception Rule

This assumes that the default rules have already been created and that an explicit proxy rule was included.

1. Navigate to SSL Orchestrator / Deployment / Interception Rules
2. Open the outbound explicit proxy rule "<sslo system name>-xp-[4/6]"
3. In the "Access Policy" field, select the "/Common/sslo\_xp\_session" policy you just created.
4. Select and add the "Common/<sslo\_system name>.app/<sslo system name>-xp-auth iRule to the rule. (if it is not already there).
5. Save the rule by clicking "Finished".

## Setting policy to bypass ICAP services

ICAP services have the ability to use LTM Policies to select which traffic is being sent to them. An example of this might be when you want to bypass ICAP for specific types of HTTP traffic, such as empty non-POST requests.

To do this, since ICAP is enabled by default, you would use the LTM policy to select the traffic that you do NOT want to be processed, and disable ICAP (by disabling the "request adapt" or "response adapt" processes).

For this use case, we want to disable all ICAP processing for traffic that is destined for our credit card processor so that sensitive traffic is not decrypted or stored in the logs of the ICAP server.

We will be disabling both request and response processing; however you can disable one or the other.

For more information on setting up ICAP policies, see

ICAP Device.

For more information on LTM Policies see the BIG-IP documentation.

### Create the LTM Policy

1. Navigate to Local Traffic / Policies
2. Create a new Draft policy by clicking Create
3. Name the policy “sslo\_skip\_cc”
4. Click “Create Policy”
5. Create a rule by clicking “Create”
6. Name the rule “sslo\_skip\_cc\_rule”
7. Add a match condition by clicking on the “+” button
8. Set the condition to “HTTP URI” “host” “is” “any of” “api.cc\_example.com” at “request” time
9. Add an action using the “+” button. Set the action to “Disable” “request adapt” at “request” time
10. Add another action.
11. Set the action to “Disable” “response adapt” at “request” time
12. Your policy should look like this:

The screenshot shows the 'General Properties' tab of an LTM Policy configuration. The 'Policy Name' is 'sslo\_skip\_cc'. The 'Name' of the rule is 'sslo\_skip\_cc\_rule'. The 'Description' field is empty. Below the properties, there is a section for 'Match all of the following conditions:'. It contains one condition: 'HTTP URI' 'host' 'is' 'any of' 'api.cc\_example.com' at 'request' time. There is an 'Options' button and an 'Add' button. Below this, there is a section for 'Do the following when the traffic is matched:'. It contains two actions: 'Disable' 'request adapt' at 'request' time, and 'Disable' 'response adapt' at 'request' time. At the bottom, there are 'Cancel' and 'Save' buttons.

13. Save the Rule

14. On the “Save Draft” button, use the down arrow to select “Save and Publish Policy”

The screenshot shows a dropdown menu for the 'Save Draft' button. The menu has two options: 'Save Draft Policy' and 'Save and Publish Policy'. The 'Save and Publish Policy' option is highlighted.

### Attach Policy to Service

1. Navigate to SSL Orchestrator / Services / ICAP Services
2. Open the ICAP service you wish to attach the policy to
3. Set the ICAP Policy field to the policy you just created.

## Configure VLANs for L2 wire Interfaces

---

Before selecting a Layer2 Network, there are some extra steps that must be taken to configure the BIG-IP. In this use case, we are configuring interfaces 1.1 and 1.2 to be part of the virtual wire. We are then adding VLAN 100 to the virtual wire interface.

### Configuration Steps:

1. Connect to the SSL Orchestrator CLI and login as root.
2. Using the following commands, create the networks.

```
(tmsh)# modify net interface 1.1 port-fwd-mode virtual-wire
modify net interface 1.2 port-fwd-mode virtual wire

(tmsh)# create net vlan l2wire-1-any tag 4096 interfaces add { 1.1 { tagged } }

(tmsh)# create net vlan l2wire-2-any tag 4096 interfaces add { 1.2 { tagged } }

(tmsh)# create net vlan-group l2wire-any-vg members add { l2wire-1-any l2wire-2-any }
mode virtual-wire

(tmsh)# create net vlan l2wire-1-100 tag 100 interfaces add { 1.1 { tagged } }

(tmsh)# create net vlan l2wire-2-100 tag 100 interfaces add { 1.2 { tagged } }

(tmsh)# create net vlan-group l2wire-100 members add { l2wire-1-100 l2wire-2-100 } mode
virtual-wire
```

## Advanced Use Cases

### Create additional outbound explicit proxy rules

(ID 740438)

By default, the system will only support one outbound explicit proxy rule. You can work around this limitation if needed by disabling SNAT and enabling address translation on additional rules.

**!** Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

#### Adding the rule

1. Navigate to SSL Orchestrator / Deployment / Interception Rules
2. Create a new Outbound Rule
3. Set options as desired, including;
  - a. Set "configuration" to "Advanced"
  - b. In the description, add "MOD:EXTRA\_XP\_OUT"
  - c. Explicit Proxy should be: "Checked"
  - d. Define the explicit proxy address and port to listen on
  - e. L7 Profile type should be: "HTTP"
  - f. L7 Profile should be set to "sslo\_<system name>-xp"
  - g. Select and add the iRule "sslo\_<system name>-xp"
  - h. Set "Access Profile" to "/Common/ssloP\_<SSLO per-req-policy>.app/ssloP\_<sslo per-req-policy>\_accessProfile"
  - i. Set "Per Request Policy" to "/Common/ssloP\_<SSLO per-req-policy>.app/ssloP\_<sslo per-req-policy>\_prpTcp"
4. Save the new rule.

#### Unlocking the application

1. Navigate to SSL Orchestrator / Deployment / Deployment Settings
2. Disable "Strict Update"
3. Add the following to the Description field: "MOD: EXTRA\_XP\_OUT"

**!** This tells F5 support that you are using the official modification for this workaround.

4. Save the application using the "finished" button.

#### Modifying the virtual server

1. Navigate to Local Traffic / Virtual Servers
2. Open the virtual server with the name "sslo\_<system name>-<rule name>"
3. Set the configuration view to "advanced"
4. Set Source Address Translation to: "None"
5. Set Address Translation to: "checked"
6. Update the Virtual Server.

## SNI for inbound listeners

---

In this use case, we have multiple HTTPS servers inside of the network that we wish to provide access to, and you wish to have SSLO select the certificate to send to the client based on SNI (Server Name Indication).

By default, an inbound rule can only support one SSL Settings group, which can only support one host. (it can support multiple certificate types per host). If you wish to configure an inbound interception rule with a network destination address instead of a single host, you probably need to configure SNI to handle selecting certificates.

We will be creating the inbound interception rule, then creating an additional Client SSL Profile to attach to it. In a production environment, you may need multiple additional Client SSL Profiles, depending on how many certificates you need to offer.

**!** Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

This assumes that you have the certificates and keys needed uploaded already on the BIG-IP.

### For more information:

- <https://devcentral.f5.com/articles/ssl-profiles-part-7-server-name-indication>
- <https://support.f5.com/csp/article/K13452>

### Create Interception Rule

1. Navigate to SSL Orchestrator / Deployment / Interception Rules
2. Create new Inbound rule
3. Name the rule "sslo\_inbound"
4. In the description, add "MOD:ADDED\_CSSL\_PRO"
5. Select and add the outside VLAN
6. Create a new SSL Settings Group by clicking on the "Create New" button in the SSL Settings field
  - a. Name the settings group "ssloT\_sslo\_inbound"
  - b. Add any of the certificates and keys.
  - c. Save the SSL Settings group
7. Set the L7 Profile Type to "HTTP"
8. Set the L7 Profile to "/Common/sslo\_<system name>-http"
9. Set "Access Profile" to "/Common/ssloP\_<SSLO per-req-policy>.app/ssloP\_<sslo per-req-policy>\_accessProfile"
10. Set "Per Request Policy" to "/Common/ssloP\_<SSLO per-req-policy>.app/ssloP\_<sslo per-req-policy>\_prpTcp"
11. Click Finished to save the rule

## Unlocking the application

1. Navigate to SSL Orchestrator / Deployment / Deployment Settings
2. Disable "Strict Update"
3. Add the following to the Description field: "MOD: INBOUND\_SNI"

! This tells F5 support that you are using the official modification for this workaround.

4. Save the application using the "finished" button.

## Create Client SSL Profiles

1. Navigate to Local Traffic / Virtual Servers
2. Find and open the virtual server for the inbound traffic in question. It should be named sslo\_<system name>-<inbound rule name>.
3. Find and note the name of the SSL Profile (Client) in use. It should be named "ssloT\_<rule name>-cssl-vhf" (note the "F" or the "T" at the end, they look alike at some resolutions).
4. Navigate to Local Traffic / Profiles / SSL / Client
5. Create additional SSL Profiles for each of the certificates that are required. In each additional Client SSL Profile, change the view to "Advanced" and override the following:
  - a. Name the profile
  - b. Set the parent Profile to the one SSLO configured. (found earlier)
  - c. The Certificate Key Chain: Select the certificate needed
  - d. If the certificate does not contain a Subject Alternative Name,
    - i. Server Name: Set to the server name
  - e. If this certificate should be the default for SNI (for when no better certificate is found)
    - i. Default SSL Profile for SNI: Set to Enabled
  - f. Save the Client SSL profile and create the next one as needed.

! Note that in creating these SSL Profiles and linking them to the existing one, you will need to manually delete or set a different parent prior to deleting the SSLO application or cleaning up the SSLO SSL Applications. Not doing this will cause SSLO to show an error when trying to delete the existing SSL profile.

## Attach Client SSL Profiles to virtual server

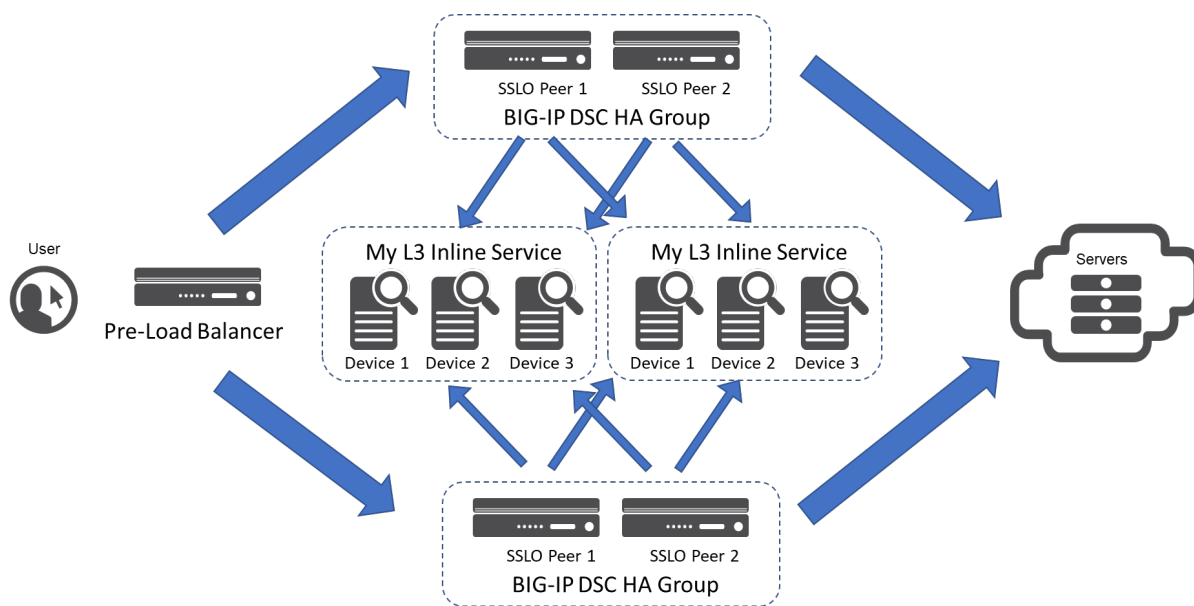
6. Navigate to Local Traffic / Virtual Servers
7. Open the inbound traffic virtual server in question.
8. Remove the existing Client SSL Profile
9. Add the newly created Client SSL Profiles.
10. Update the Virtual Server

## Using SSLO in a horizontally scaled architecture

ID 740440

In this use case, you have chosen to deploy SSLO in a horizontally scaled architecture, one where there is a pre-load balancer, which balances traffic across multiple SSL Orchestrator systems. By default, SSL Orchestrator does not support this approach to scaling if the systems will share Layer 3 or HTTP services, however, with some modifications, it can work.

The issue is that if multiple SSLO devices are sending traffic to a Layer 3 or HTTP service device, the service devices will have no way of knowing how to route the traffic back to the correct SSLO system. The workaround for this is to enable SNAT on the service virtual servers so that the source address can be used to determine which system the traffic comes from. Then configure the service device to route traffic back to the correct SSLO using policy-based routing.



To deploy this type of horizontally scaled architecture, the high-level steps you would follow are:

1. Deploy and configured the first SSLO system (including HA pair / standby system)
2. Deploy and configure the second SSLO system, for this one you want to disable “auto-manage” on any Layer 3 and HTTP services, and setup manual addressing for these. You will want to make sure that you use addresses that are not in use elsewhere in the system (take a look at the Network / Self IP addresses to understand what is in use)
3. On both devices, disable the strict update and enable SNAT on the Layer 3 and HTTP services.
4. Configure the service devices to correctly route traffic to the correct SSLO using policy-based routing. (See the documentation for the service device to determine how to configure this).



- ! Note: Using this approach, the source address for the traffic will now be the SSLO SNAT address, not the original address. This may impact how the service device handles and reports on the intercepted traffic.
- ! If the monitoring devices require the original source address, consider adding a “X-Forwarded-For” HTTP header with the source address. (see Add HTTP Headers to traffic)
- ! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

### Enabling SNAT for service virtual server

1. Navigate to SSL Orchestrator / Services
2. Open each of the Layer 3 and HTTP services, on each;
  - a. Disable strict update
  - b. Add the text “FC:ADD SNAT” to the description
  - c. Note the IP network(s) that is(are) configured for the “To Service”
  - d. Save the service
3. Navigate to Local Traffic / Address Translation / SNAT Pool List
4. Create a SNAT pool for each of the Layer 3 and HTTP services.
  - a. Name the SNAT pool based on the service
  - b. Use IP address ranges within the network in each of the services noted earlier.
  - c. The addresses should start AT LEAST 3 addresses in from the first available address in the network range.
  - d. Verify that any addresses selected are not present in the Network / Self IPs list
  - e. Verify that any addresses selected are not used by service devices or other systems in that network.
  - f. Save the SNAT Pool
5. Navigate to Local Traffic / Virtual Servers
6. Locate the sending virtual servers for Layer 3 and HTTP services. These will be named: “ssloS\_<service name>-t” and “ssloS\_<service name>-u”.
7. For each of these, do the following:
  - a. Open the virtual server.
  - b. Enable SNAT and select the correct SNAT pool.
  - c. Save the virtual server

# Provide IP address persistence for outbound traffic

---

ID 740443

By default, SSL Orchestrator does not maintain IP address persistence for outbound traffic, this is not a problem for most cases, however in some rare instances it can cause a problem. For example;

If a user goes to a site that performs federated authentication and tracks the client IP address in the federation process. When redirected from the authentication system to the application, if the client IP changes, the user connection may be blocked.

In some cases, simply only defining one SNAT address in SSLO could address this, however there is a limit to the number of connections that a single SNAT address can support. If the number of concurrent connections is expected to be above 64,000, then the below workaround could be considered.

## What it does:

On initialization, this workaround will count the number of SNAT addresses and save that in a variable.

Then, on receiving traffic, it takes either the client IP address, or both the client IP address and source port, and generates a hashed value that is used to select the outbound source address. This is done using the following process:

Example based on:

8. List of SNAT addresses:
  - 10.1.1.1,
  - 10.1.1.2,
  - 10.1.1.3;
9. Traffic source IP: 10.13.1.23
1. Generate a crc32 value from the source information (client IP address or client IP address AND source port)
  - a.  $10.13.1.23 = 0x7C99BA0F$
2. Divide that value by the number of SNAT addresses and get the reminder
  - a.  $0x7C99BA0F \text{ modulo } 3 = 1$
3. Use that remainder as the zero offset to select the SNAT address.
  - a. Address offset 1 = 10.1.1.2

This will insure that traffic that always uses the same source IP and port will always use the same SNAT address.

**!** Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

## Setup the iRules

1. Install the SSLO Extras (Using the SSLO Extras Library)
2. Navigate to Local Traffic / iRules
3. Create a new rule called “sslo\_persistent\_snat”
4. In the definition field, add the following text:

```
# FC: create persistence in SNAT

when RULE_INIT { call /Common/sslo_extras/snatch_pool_perc_init "sslo_persistent_snatch" }

# Uncomment ONE of the following based on what you would like to base persistence on.
# when HTTP_REQUEST { call /Common/sslo_extras/snatch_perc_client_addr }
# when HTTP_REQUEST { call /Common/sslo_extras/snatch_perc_client_addr_source_port }
```

5. Uncomment one of the last two lines, depending on which method you wish to use for persistence.
6. Save the rule.

## Setup the SNAT Pool

9. Navigate to Local Traffic / Address Translation / SNAT Pool List
10. Create a SNAT pool named “sslo\_persistent\_snatch”
11. Add the SNAT addresses you wish to use to the pool
12. Save the pool by clicking “Finished”

## Apply the rules to the virtual servers

13. Navigate to SSL Orchestrator / Deployment / Interception Rules
14. Open the rule “<sslo system name>-in-t”
15. Add the iRule you created to this rule in the “resources” tab.
16. If present, repeat for the rule “<sslo system name>-in-u”.

# Enable Client Certificate Constrained Delegation for SSLO

ID 740971

SSLO 4.0 does not provide options to configure Client Certificate Constrained Delegation for inbound traffic. This would be useful for customers that require inbound inspection for applications requiring client certificate authentication. C3D requires specially-formatted client and server SSL profiles and a CA certificate and private key to forge external client certificates to internal services.

## For more information:

- [https://support.f5.com/kb/en-us/products/big-ip\\_ltm/manuals/product/bigip-ssl-administration-13-1-0/4.html](https://support.f5.com/kb/en-us/products/big-ip_ltm/manuals/product/bigip-ssl-administration-13-1-0/4.html) (see “About client certificate constrained delegation”)

This use case assumes you have SSLO configured and the inbound interception rule defined.

! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

## Unlocking the application

1. Navigate to SSL Orchestrator / Deployment / Deployment Settings
2. Disable "Strict Update"
3. Add the following to the Description field: "MOD:ENABLE\_C3D"

**!** This tells F5 support that you are using the official modification for this workaround.

4. Save the application using the "finished" button.

## Create a new Client SSL Profile

1. Navigate to Local Traffic / Virtual Servers
2. Find and open the virtual server for the inbound traffic in question. It should be named `sslo_<system name>-<inbound rule name>`.
3. Find and note the name of the SSL Profile (Client) in use. It should be named `"ssloT_<rule name>-cssl-vhf"` (note the "F" or the "T" at the end, they look alike at some resolutions).
4. Navigate to Local Traffic / Profiles / SSL / Client
5. Create the Client SSL Profile with minimally the following items:
  - a. Name the profile
  - b. Set the parent Profile to the one SSLO configured. (found earlier)
  - c. Configuration
    - i. Certificate Key Chain: Select the server certificate and private key that clients will see.
  - d. Client Authentication
    - i. Client Certificate: Require.
    - ii. Trusted Certificate Authorities: Select a CA certificate or CA bundle to allow the BIG-IP to properly trust-validate external client certificates.
  - e. Client Certificate Constrained Delegation
    - i. Client Certificate Constrained Delegation: set to Enabled.
    - ii. If you are using OCSP validation:
      1. OCSP: Select or create the OCSP profile.
      2. Unknown OCSP Response Control: Define this to control what happens if OCSP validation returns an unknown status.
  - f. Save the Client SSL profile and create the next one as needed.
6. Save the profile

## Create a new Server SSL Profile

1. Navigate to Local Traffic / Virtual Servers
2. Find and open the virtual server for the inbound traffic in question. It should be named `sslo_<system name>-<inbound rule name>`.
3. Find and note the name of the SSL Profile (Client) in use. It should be named `"ssloT_<rule name>-sssl-vhf"` (note the "F" or the "T" at the end, they look alike at some resolutions).
4. Navigate to Local Traffic / Profiles / SSL / Server
5. Create the Client SSL Profile with minimally the following items:
  - a. Name the profile
  - b. Set the parent Profile to the one SSLO configured. (found earlier)
  - c. Configuration

- i. Certificate/Key – this certificate and key are used by C3D as the base private key and information to forge new client certificates. It's okay to select the built-in default.crt/.key here.
- d. Server Authentication
  - i. Server Certificate – set to require.
  - ii. Expire and Untrusted Certificate Response Control – typically set these to ignore unless you intend to also import a CA or CA bundle to validate internal server certificates.
  - iii. Frequency – set to always to disable server-side certificate caching.
- e. Client Certificate Constrained Delegation:
  - i. Client Certificate Constrained Delegation – set to Enabled.
  - ii. CA Certificate/Key – this CA certificate and private key are used to re-sign (forge) client certificates to the internal servers.
- 6. Save the profile

### **Attach Client SSL Profiles to virtual server**

1. Navigate to Local Traffic / Virtual Servers
2. Open the inbound traffic virtual server in question.
3. Replace the existing Client SSL Profile with the one you just created
4. Replace the existing Server SSL Profile with the one you just created
5. Update the Virtual Server

## **Add HTTP Headers to traffic**

---

Due to how SSLO handles traffic, using a HTTP Headers action in an APM Per-Request policy will cause traffic to skip any inline services such as Layer 2, Layer 3, and HTTP services. The workaround for this is to use an iRule to add the headers instead of a VPE agent.

Depending on when you wish the headers to be added, there are different approaches to doing this, in general, the three major approaches are;

1. Adding a header to all traffic upon entering the system
2. Adding a header to traffic to selected traffic upon entering the system
3. Adding a header only to traffic going to a specific service (and removing it before forwarding the traffic)

**!** Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

## Common first steps

For all of these operations, there are a few common first steps.

7. Setup the SSLO Extras Library (see Using the SSLO Extras Library)
8. Navigate to Local Traffic / iRules
9. Create an iRule and name it “sslo\_add\_header”
10. If you are adding a simple string for the header, you will:
  - a. In the description field enter the following iRule, replacing <header name> and <header value> with the appropriate values. (note, you can pass multiple values to this procedure by simply adding more header\_name and header\_value strings.)

```
when CLIENT_ACCEPTED {  
    call /Common/sslo_extras/add_headers <header_name> <header_value>  
    [<header_name> <header_value> ...]  
}
```

! If your value is based on a TCL variable, remember to add code to get the value, and to add the “\$” before the variable name.

! If you are entering strings, remember to enclose them with double quotes “ ”.

- b. So, your final iRule should look somewhat like:

```
when CLIENT_ACCEPTED {  
    call /Common/sslo_extras/add_headers “X-Custom-Header” “myValue”  
}
```

11. If you wish to add the username to the header (for example, for an “X-Authenticated-User”, you would:

- a. In the description field enter the following iRule:

```
when CLIENT_ACCEPTED { call /Common/sslo_extras/add_auth_header }
```

This assumes that authentication is setup and configured and will add the header “X-Authenticated-User” with a value of the username from the authentication process.

12. Save the iRule

13. If you also want this header removed:

- a. Create another iRule called “sslo\_rem\_header”
  - b. In the definition field, add the following after replacing <header\_name> with the name of the header:

```
when CLIENT_ACCEPTED {  
    call /Common/sslo_extras/rem_headers <header_name>  
}
```

- c. Save the iRule

## Add Header for all traffic

This use case would be where you wished to add a header to all of the HTTP and HTTPS traffic in a system.

1. Navigate to SSL Orchestrator / Deployment / Interception Rules
2. Open the rule “sslo\_<sslo system name>-in-t”
3. Add the “sslo\_add\_headers” iRule to the rule in the resources / iRules field..

## Add Header for Selected Traffic

This use case would be where you wished to add a header to selected HTTP and HTTPS traffic as it transit's the system. An example of a place this could be helpful is in adding a header to traffic going to a specific domain to control application access.

Some common headers that could fit this use case:

<b>Google Apps</b>	<p>To limit access to a specific domain (<a href="https://support.google.com/a/answer/1668854?hl=en">https://support.google.com/a/answer/1668854?hl=en</a>)</p> <p><b>Filter for domain:</b> 10. google.com</p> <p><b>Header Text:</b> “X-GoogApps-Allowed-Domains”</p> <p><b>Header Value:</b> A space separated list of google and customer domains that the user should be able to access</p>
<b>YouTube Videos</b>	<p>To limit access to restricted mode (<a href="https://support.google.com/a/answer/6214622?hl=en">https://support.google.com/a/answer/6214622?hl=en</a>)</p> <p><b>Filter for domain:</b> 11. www.youtube.com 12. m.youtube.com 13. youtubei.googleapis.com 14. youtube.googleapis.com 15. www.youtube-nocookie.com</p> <p><b>Header Text:</b> "YouTube-Restrict"</p> <p><b>Header Value:</b> Either “Strict” or “Moderate” (depending on your goal, see <a href="https://support.google.com/a/answer/6212415">https://support.google.com/a/answer/6212415</a> for more information)</p>

---

**Microsoft Office 365 /  
Azure**

To limit access to an account  
(<https://docs.microsoft.com/en-us/azure/active-directory/manage-apps/tenant-restrictions>)

**Filter for domain:**

There are a number of domains that Microsoft uses, and these change regularly, see <https://docs.microsoft.com/en-us/office365/enterprise/urls-and-ip-address-ranges>), these would have to be added to a custom url category.

**Header Text:**

“Restrict-Access-To-Tenants” or “Restrict-Access-Context”  
(depending on how you wish to restrict access (see MS link above for more information)

**Header Value:**

If restricting by tenants: this is a comma separated of tenant domains that can access. (i.e.  
“contoso.onmicrosoft.com,fabrikam.onmicrosoft.com”)

If restricting by context, this is a single directory ID that is allowed access. (i.e. “456ff232-3512-5h23-b3b3-3236w0826f3d”)

(see above link for more information on these)

---

In this example use case, we will filter based on the URL, assuming that the URL(s) to filter on have already been added to a custom category called “add\_headers\_cat” (you would probably name it based on the type of application you are handling)

! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

**Setting up the policy**

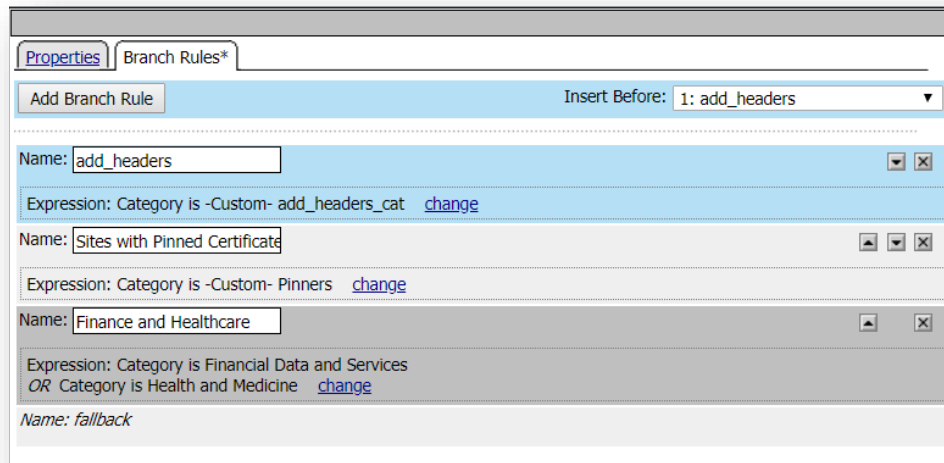
Once you have the categories setup you need, you will need to setup the policy.

In this case, since we know that the traffic going to these will all be intercepted, and that we do not need additional handling, we are simply adding another branch off of the “Category Branching” agent, however, in another case we might want to add this to a new macro before the “SSL Intercept Policy” macro so that the resulting traffic is still run through all of the branches in that macro.

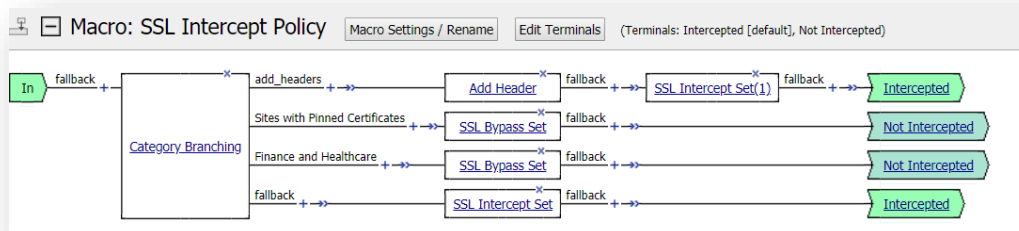
! If you wish to also categorize HTTP (non-encrypted) traffic, you will need to enable HTTP categorization (See Categorize URLs for HTTP (non-encrypted) traffic)



1. Navigate to SSL Orchestrator / Policies / Per-Request Policies
2. Find the Per-Request Policy that you need to modify and show all APM Per request policies using the “+ Show All” button.
3. Select the “ssloP\_<chain name>\_prpTcp” APM Per Request Policy.
4. Expand the “SSL Intercept Policy” macro.
5. Edit the “Category Branching” agent and select the “Branch Rules” tab.
6. Click on [Add Branch Rule] and name the rule “add headers”
7. Click on the “change” link next to “Expression: Empty”
8. Click on [Add Expression]
9. Set the Agent Sel: and Condition to “Category Lookup”
10. Select the category “-Custom- add\_headers\_cat”
11. Save the expressions
12. Your agent should now look something like:



13. On the “add headers” branch, add an iRule Event agent and name it “Add Header”
14. Set the ID to “sslo\_add\_header” and the expected data to “Client Accepted”
15. Save the agent.
16. Since the traffic still needs to be intercepted, we need to add the SSL Intercept Set agent to the branch as well by clicking on the “+” after the “Add Header” event.
17. Save the Intercept Set agent.
18. Make sure that this branch terminates in an “Intercepted” terminal.
19. Your SSL Intercept Policy macro should now look like this:



## Add and remove header for a service

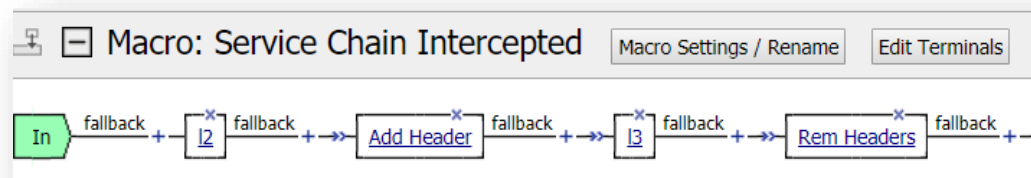
In some cases, a customer will wish to inject a HTTP header value only for a specific service. For example, for sending an X-Authenticate-User header to non-HTTP service device. Then, remove the header on return from the service device. The following is an example of how this can be done.

Note this can already be done for HTTP services by selecting the Authentication Offload option when setting up the service, however that option is not available in other services.

This assumes that you have setup both of the iRules listed in the common first steps, using the iRule that adds the authenticated user for the “sslo\_add\_header” iRule.

! Note: Before implementing this use case, please read and understand the Modification of SSLO deployed objects

1. Navigate to SSL Orchestrator / Policies / Per-Request Policies
2. Find the Per-Request Policy that you need to modify and show all APM Per request policies using the “+ Show All” button.
3. Select the “ssloP\_<chain name>\_prpTcp” APM Per Request Policy.
4. Find and expand the “Service Chain Intercepted” Macro
5. Find the service that you wish to send the header to.
6. Immediately before the service, add an iRule Event agent and call it “Add Headers”
7. Set the ID to “sslo\_add\_header” and the Expect data field to “Client Accepted”
8. Immediately after the server, add an iRule Event agent and call it “Rem Headers”
9. Set the ID to “sslo\_rem\_header” and the Expect data field to “Client Accepted”
10. Your Service Chain Intercepted Macro should look somewhat like:  
(depending on the services that you have configured)



## Configuring external communications for services

---

Best practice for architecting a secure SSL Orchestrator system requires that the service devices should not have any access outside of their protected networks. (see SSL Orchestrator v4.0 Architecture guide) However in many cases the service devices will require it for validation, software updates, or other reasons.

In these cases, rather than simply providing a route out of the protected network, F5 recommends creating another outbound rule for that service device and where possible locked down to the specific end addresses that it needs to communicate with.

The steps for this are:

1. Navigate to SSL Orchestrator / Deployment / Interception Rules
2. Create a new outbound rule
3. Provide a name for the rule (we suggest something like <service name>\_to\_<destination>)
4. Set the protocol as needed
5. Set the Source address to the IP address of the service device
6. Set the Destination address and service port to the IP address and port of the expected destination.
7. Select the VLAN that the service is on.
8. Leave the remainder of the settings at the default of “None” or unchecked.

# Appendixes

## iRule: sslo\_extras

---

```
## SSL0 Extras library
## Version 4.0.1

# Copied from iRule_lib.tcl for consistent logging
# emit a log message
# uses ACCESS::log ssl-orchestrator.[debug|notice|error] $msg to make use of
# the Common Logging Framework

# Usage:
# call log [0|1|2] "log msg"
proc fc_log {level msg} {
    if {$msg eq ""} { return }

    if {$level == 0} {
        ACCESS::log ssl-orchestrator.error "[virtual] $msg"
    } elseif {$level == 1} {
        ACCESS::log ssl-orchestrator.notice "[virtual] $msg"
    } elseif {$level == 2} {
        ACCESS::log ssl-orchestrator.debug "[virtual] $msg"
    }
} ; #proc log

proc fc_log_error {msg} {
    call fc_log 0 $msg
}

proc fc_log_notice {msg} {
    call fc_log 1 $msg
}

proc fc_log_debug {msg} {
    call fc_log 2 $msg
}

## *****
##
## Rules for HTTP Traffic not categorized issue
##
## *****

# The below rules depend on a datagroup of url-categories existing, the below scripts can be used to
# create that datagroup.
# This one will create the group initially:

# tmsh list sys url-db url-category |grep -E 'sys url-db url-category|    cat-number' |sed -e 's/sys url-
db url-category //g;s/    cat-number //g;s/ {//g' |sed "N;s/\n/ := /" |sed -e 's/$/,/' >
/var/tmp/urlcat.txt
# tmsh create /sys file data-group urlcat_file separator "==" source-path file:/var/tmp/urlcat.txt type
string
# tmsh create /ltm data-group external urlcat_dg external-file-name urlcat_file

# This one will modify it after creation:

# tmsh list sys url-db url-category |grep -E 'sys url-db url-category|    cat-number' |sed -e 's/sys url-
db url-category //g;s/    cat-number //g;s/ {//g' |sed "N;s/\n/ := /" |sed -e 's/$/,/' >
/var/tmp/urlcat.txt
# tmsh modify /sys file data-group urlcat_file separator "==" source-path file:/var/tmp/urlcat.txt
```

```

# tmsl modify /ltm data-group external urlcat_dg

## This event is used to capture the HTTP Host header for HTTP category lookup
## This sets parameter "perflow.scratchpad" with the category ID
## This assumes a datagroup called urlcat, but can be overridden at call time.

# This iRule performs the category lookup, parses the data group, and returns the matching values as the
# required string of delimited category number values. The value is temporarily stored in a scratchpad
# perflow variable.

## Usage:
## call /Common/sslo_fc_lib/get_urlcat <optional - name of datagroup with cats>

proc get_urlcat args {
    switch [ACCESS::perflow get perflow.irule_agent_id] {
        "http_data_collector" {
            call log_fc_debug "FC_get_urlcat: Looking up HTTP category ID"
            if {[llength $args] > 0} {
                set urlcat_dg [lindex $args 0]
            } else {
                set urlcat_dg "urlcat"
            }
            call log_fc_debug ["FC_get_urlcat: Setting datagroup to $urlcat_dg"]

            if { [class exists ${urlcat_dg}] } {
                if { [ACCESS::perflow get perflow.scratchpad] ne "" } {
                    set URL "http://[ACCESS::perflow get perflow.scratchpad]"
                    call log_fc_debug ["FC_get_urlcat: checking urlcat for url: $URL"]
                    set res [CATEGORY::lookup ${URL} request_default_and_custom]
                    set found 0
                    set cat_id ""
                    foreach x ${res} {
                        set y [string map {"Common/" ""} ${x}]
                        set srch [class lookup ${y} urlcat]
                        if { ${srch} ne "" } {
                            if { !${found} } {
                                set cat_id "| ${srch} |"
                                set found 1
                            } else {
                                append cat_id " ${srch} |"
                            }
                        }
                    }
                }
                if { ${found} } {
                    ACCESS::perflow set perflow.scratchpad ${cat_id}
                    call log_fc_debug ["FC_get_urlcat: returning cat_id: $cat_id"]
                } else {
                    call log_fc_debug "FC_get_urlcat: no matching cat_id"
                }
            } else {
                call fc_log_error ["FC_get_urlcat: Datagroup $urlcat does not exist"]
            }
        }
    }
}

## HTTP Host collect for HTTP category lookup
# It should also be noted that if iRule event agent in this position (in the path)

```

```

# is set to expect data at CLIENT_ACCEPTED, the HTTP Host header is not available.
# If set to expect data at HTTP_REQUEST, the HTTP Host header is available and can
# be used to perform a category query, but then SSLO does not send any traffic through
# security services. To work around this, a small modification is required in the -in_t
# iRule, around line 141, to capture the HTTP Host header and store it in the
# perflow.scratchpad variable, so that the iRule event can work at CLIENT_ACCEPTED.
# The added code in the _in_t rule extracts the HTTP Host header from the payload if
# this is client_speaks_first traffic [ctx(csrf)] and is detected HTTP. The value is
# then injected into a temporary perflow variable. This scratchpad variable feeds
# the http_data_collector iRule above. Modification in red below.

## Usage:
## call /Common/sslo_fc_lib/store_host_header $said

proc store_host_header said {
    if {[regexp {[A-Z]+} (?:/|https?:/[A-Za-z\d-]{1,63}[.]|\\[[a-zA-Z\d:]+\\]|(?:[*]\x20))} $said
    junk method] &&
        ([lsearch -exact -sorted {CONNECT DELETE GET HEAD LOCK OPTIONS PATCH POST PROPFIND PUT
TRACE UNLOCK} $method] >= 0)} {
        set host_hdr [findstr ${said} "Host: " 6 "\r\n"]
        if { ${host_hdr} ne "" } {
            ACCESS::perflow set perflow.scratchpad ${host_hdr}
            call log_fc_debug "FC_Store_host_header: Setting host header to $host_hdr"
        } else {
            call log_fc_debug "FC_Store_host_header: Empty host header found"
        }
    } else {
        call log_fc_debug "FC_Store_host_header: Not HTTP or HTTPS, or unable to parse header."
    }
}

}

## *****
##
## Rules for adding HTTP Headers in SSLO
##
## *****

## This will add headers to a http conversation.
## This expects a list of header value header value ...
## and will throw an error if traffic passed is not a multiple of 2
## this should be added to either the in_t_virtual server, or the _t_virtual server for the service.

## Usage:
## call /Common/sslo_fc_lib/add_headers <header_name> <header_value> [<header_name> <header_value> ...]

proc add_headers add_header_list {
    sharedvar ctx
    call log_fc_debug "FC_add_headers: Called add_headers"
    if {$ctx(ptcl) eq "http" || $ctx(ptcl) eq "https"} {
        if {[expr {[llength $add_header_list] % 2}] == 0} {
            HTTP::enable
            foreach {h_name h_value} $add_header_list {
                HTTP::header insert $h_name $h_value
                call log_fc_debug ["FC_add_headers: Adding Header: $h_name = $h_value"]
            }
        } else {
            call log_fc_debug "FC_add_headers: Invalid number of headers passed"
            HTTP::disable
        }
    } else {

```

```

        call log_fc_debug "FC_add_headers: Not HTTP or HTTPS"
        HTTP::disable
    }
}

## This will add the auth headers to a http conversation.
## this should be added to the _t virtual server for the service.

## Usage:
## call /Common/sslo_fc_lib/add_auth_header
proc add_auth_header {
    call log_fc_debug "FC_add_auth_header: Called add_auth_header"
    if { [ACCESS::session exists] } {
        set sessionID [ACCESS::session sid]
        set user [ACCESS::session data get "session.logon.last.username"]
        call add_headers "X-Authenticated-User" $user
    } else {
        call log_fc_debug "FC_add_auth_header: No session exists"
    }
}

## This will add headers to a http conversation.
## This expects a domain, then a list of header value header value ...
## and will throw an error if traffic passed is not a multiple of 2
## this should be added to either the in_t_ virtual server, or the _t virtual server for the service.

## Usage:
## call /Common/sslo_fc_lib/add_headers_if_in_domain "<domain_name>" <header_name> <header_value>
[<header_name> <header_value> ...]
proc add_headers_if_in_domain {in_domain args} {
    set domainhost [string tolower [HTTP::host]]
    call log_fc_debug "FC_add_headers_if_in_domain: Called add_headers_if_in_domain"
    if {[${domainhost} equals $in_domain] } {
        call add_headers $args
    } else {
        call log_fc_debug ["FC_add_headers_if_in_domain: Host ${domainhost} not in domain $in_domain"]
    }
}

## This will add headers to a http conversation.
## This expects a domain, then a list of header value header value ...
## and will throw an error if traffic passed is not a multiple of 2
## this should be added to either the in_t_ virtual server, or the _t virtual server for the service.

## Usage:
## call /Common/sslo_fc_lib/add_headers_if_in_domain_datagroup "<datagroup_name>" <header_name>
<header_value> [<header_name> <header_value> ...]
proc add_headers_if_in_domain_datagroup {in_domain args} {
    set domainhost [string tolower [HTTP::host]]
    call log_fc_debug "FC_add_headers_if_in_domain_datagroup: Called add_headers_if_in_domain_datagroup"
    if { [class match $domainhost eq $in_domain] } {
        call add_headers $args
    } else {
        call log_fc_debug ["FC_add_headers_if_in_domain_datagroup: domain $domainhost not in datagroup
$in_domain"]
    }
}

## This will remove headers to a http conversation.
## This expects a list of headers
## this should be added to the _D virtual server for the service.

```

```

## Usage:
## call /Common/sslo_fc_lib/rem_headers <header_name> [<header_name> ...]

proc remove_headers {rem_header_list} {
    sharedvar ctx
    call log_fc_debug "FC_rem_headers: Called rem_headers"
    if {$ctx(ptcl) eq "http" || $ctx(ptcl) eq "https"} {
        HTTP::enable
        foreach h_name $rem_header_list {
            HTTP::header remove $h_name
            call log_fc_debug ["FC_rem_headers: Removing Header: $h_name"]
        }
    } else {
        HTTP::disable
        call log_fc_debug "FC_rem_headers: Not HTTP or HTTPS"
    }
}

## *****
##
## Rules for providing persistence to outbound traffic
##
## *****

# Based on this article from devcentral
# https://devcentral.f5.com/codeshare/snat-pool-persistence?lc=1

# usage: (add to RULE_INIT)
# call /Common/sslo_fc_lib/snat_pool_perc_init "<pool_name>"

proc snat_pool_perc_init {pool_name} {

    set static::snatpool_name $pool_name
    set static::members_cmd "members -list $static::snatpool_name"
    unset -nocomplain static::snat_ips
    set static::i 0
    foreach static::snat_ip [eval $static::members_cmd] {
        set static::snat_ips($static::i) [lindex $static::snat_ip 0]
        incr static::i
    }

    set static::array_size [array size static::snat_ips]
    call fc_log_notice "Loaded $static::array_size SNAT IPs from $static::snatpool_name: [array get static::snat_ips]"

    unset static::snatpool_name static::members_cmd static::i static::snat_ip
}

# Select one fo the two persistence methods below and create a rule with event HTTP_REQUEST:

# this one will hash based on client address

# usage:
# call /Common/sslo_fc_lib/snat_perc_client_addr

proc snat_perc_client_addr {
    snat $static::snat_ips([expr {[crc32 [IP::client_addr]] % $static::array_size}])
}

# This one will hash based on client address and remore port

```



```
# usage:
# call /Common/sslo_fc_lib/snat_perc_client_addr_source_port
proc snat_perc_client_addr_source_port {
    snat $static::snat_ips([expr {[crc32 [IP::client_addr][TCP::remote_port]] % $static::array_size}))
}
```

# Legal Notices

## Trademarks

AAM, Access Policy Manager, Advanced Client Authentication, Advanced Firewall Manager, Advanced Routing, AFM, APM, Application Acceleration Manager, Application Security Manager, Applications without Constraints, ARX, AskF5, ASM, BIG-IP, BIG-IP EDGE GATEWAY, BIG-IQ, BIG-IP iControl, Cloud Extender, CloudFucious, Cloud Manager, Clustered Multiprocessing, CMP, COHESION, Data Manager, DDoS Frontline, DDoS Hybrid Defender, DDoS SWAT, Defense.net, DevCentral, DevCentral [DESIGN], DNS Express, DSC, DSI, Edge Client, Edge Gateway, EDGE MOBILE, EDGE MOBILITY, EdgePortal, ELEVATE, EM, Enterprise Manager, ENGAGE, F5, F5 Agility, F5 iApps, F5[DESIGN], F5 Certified [DESIGN], F5 iControl, F5 LINK CONTROLLER, F5 Networks, F5SalesXchange [DESIGN], F5Synthesis, f5Synthesis, F5Synthesis[DESIGN], F5 TechXchange [DESIGN], F5 TMOS, Fast Application Proxy, Fast Cache, FCINCO, Global Traffic Manager, GTM, GUARDIAN, Herculon, iApps, IBR, Intelligent Browser Referencing, Intelligent Compression, IPv6 Gateway, iCall, iControl, iHealth, iQuery, iRules, iRules OnDemand, iSeries, iSession, L7 RateShaping, LC, Link Controller, Local Traffic Manager, LTM, LineRate Operating System, LineRate Point, LineRate Precision, LineRate Systems [DESIGN], LROS, LTM, Message Security Manager, MSM, OneConnect, Packet Velocity, PEM, Policy Enforcement Manager, Protocol Security Manager, PSM, Real Traffic Policy Builder, Ready Defense, SalesXchange, ScaleN, Signalling Delivery Controller, Silverline, Silverline Threat Intelligence, SDC, SSL Acceleration, SSL Everywhere, SSL Orchestrator, SDAC (except in Japan), StrongBox, SuperVIP, SYN Check, SYNTHESIS, TCP Express, TDR, TechXchange, TMOS, TotALL, Traffic Management Operating System, Traffix Systems, Traffix Systems (DESIGN), Transparent, Data Reduction, UNITY, VAULT, vCMP, VE F5 [DESIGN], Versafe, Versafe [DESIGN], VIPRION, Virtual Clustered Multiprocessing, WAF Express, WebSafe, We Make Apps Go [DESIGN], We Make Apps GO, and ZoneRunner, are trademarks or service marks of F5 Networks, Inc., in the U.S. and other countries, and may not be used without express written consent.

All other product and company names herein may be trademarks of their respective owners.

## Patents

This product may be protected by one or more patents. See the F5 Patents page (<https://www.f5.com/about/guidelines-policies/patents>).

## Notice

THE SOFTWARE, SCRIPTING, AND COMMAND EXAMPLES ARE PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES, OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE, SCRIPTING AND COMMAND EXAMPLES, OR THE USE OR OTHER DEALINGS WITH THE SOFTWARE, SCRIPTING, AND COMMAND EXAMPLES.

## Publication Date

This document was published in Sept 2018.

## Copyright

Copyright © 2013-2018, F5 Networks®, Inc. All rights reserved.

F5 Networks, Inc. (F5) believes the information it furnishes to be accurate and reliable. However, F5 assumes no responsibility for the use of this information, nor any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent, copyright, or other intellectual property right of F5 except as specifically described by applicable user licenses. F5 reserves the right to change specifications at any time without notice.