Firepower System Remediation API Guide

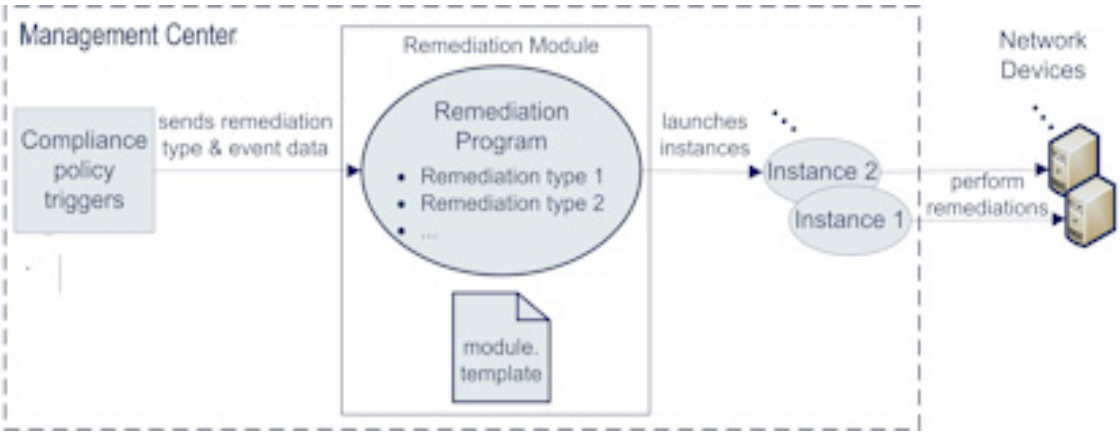
Cisco Systems, Inc.
www.cisco.com

Cisco has more than 200 offices worldwide.
Addresses, phone numbers, and fax numbers
are listed on the Cisco website at
www.cisco.com/go/offices.

Version 6.0
September 13, 2018

Understanding the Remediation Subsystem

The Firepower System® remediation API allows you to create remediations that your Firepower Management Center can automatically launch when conditions on your network violate the associated correlation policy. *A remediation is* the response your software program executes to mitigate the detected condition. For example, you can block traffic at a router on the source or destination IP address, or initiate a host Nmap scan to assess the host status. If multiple rules in a policy trigger, the Firepower Management Center can launch responses for each rule. A *remediation module* is the package of files you install on the Firepower Management Center to perform the response. A remediation module can incorporate several *remediation types* as shown in the graphic below.



In the example shown here, one of the system-provided remediation modules, the Cisco PIX router module, performs two remediation types: it either blocks packets by source IP address or blocks them by destination IP address.

If a remediation module targets multiple devices on your network (routers, hosts, and so forth), you configure your remediation module to perform multiple *instances*, one per device, when the correlation policy triggers. An instance is an instantiation of the remediation module, with one or more remediation types that correspond to functions in the remediation module code, and with a set of variables needed to run on the target device. For each instance, you specify the remediation type or types it executes and the instance-specific information such as the device's IP address and password for the remediation to access the target device on your network.

The Firepower System's domains feature allows you to implement multitenancy within your deployment by segmenting user access to managed devices, configurations, and events. For systems that have more than one domain, users can create remediations for different domain levels. A remediation is not visible to ancestor or parallel domains from the one in which it was created. While a user in a child domain can view a remediation created in their ancestor domain, they cannot modify or delete it. A remediation created at the root domain can be viewed and used by all child domains.

Prerequisites

Before using the remediation API for custom remediations, you should be familiar with information in the following categories:

Firepower System

To understand the information in this guide, you should be familiar with the features and nomenclature of the Firepower System, and the functions of certain components:

- the Firepower Management Center role in the Firepower System architecture
- correlation policy management module on the Firepower Management Center
- remediation management module on the Firepower Management Center

See the *Firepower System User Guide* for further information.

Programming Requirements and Support

You must be able to code your custom remediation in Perl or shell script, or as a precompiled, statically-linked C program (with the exception of links to routines in glibc).

In addition, you must be able to produce a configuration file in XML for each remediation module. This file is called `module.template`. See the system-provided remediation modules for samples of this file. For module locations on the Firepower Management Center, see Understanding the Remediation Subsystem File Structure, page 4-4.

For each instance you add, the Firepower Management Center generates an instance-specific XML configuration file called `instance.conf`. Your code must parse this file each time a remediation instance executes.

The following table lists the packages available on the Firepower Management Center as resources for writing and executing your remediation program.

Table 1-1 Additional Packages

Additional Packages	Location
GNU bash, version 3.2.33(1)-release	`/bin/bash`
tcsh 6.17.00	`/bin/tcsh`
glibc 2.7	`/lib/libc-2.7.so`
perl v5.10.1	`/usr/bin/perl`
Net::Telnet	`N/A`
Net::SSH::Perl	`N/A`
XML::Smart	`N/A`

Cisco-Provided Remediation Modules

The following table describes the predefined remediation modules included with the Firepower Management Center. You should use these modules for reference when designing your remediation programs.

The system-provided modules are already installed on the Firepower Management Center and include both the remediation executable (in Perl and C) and completed `module.template` configuration file for each module. For information on the easy steps to deploy system-provided remediation modules, see the *Firepower System User Guide*.

Table 1-2 Cisco-Provided Remediation Modules

Module Name	Function
Cisco IOS Null Route	if you are running Cisco routers that use Cisco IOS® Version 12.0 or higher, allows you to dynamically block traffic sent to an IP address or network that violates a correlation policy
Cisco pxGrid Mitigation	performs a pxGrid mitigation against the involved IP addresses replaces the PIX Shun module
Nmap Scanning	allows you to actively scan specific targets to determine operating systems and servers running on those hosts
Set Attribute Value	allows you to set a host attribute on a host where a correlation event occurs
FMC Remediation Module for ACI	quarantines endpoint using the Application Policy Infrastructure Controller not pre-installed, download from https://software.cisco.com/download/home/286259687/type
FMC Remediation Module for Tetration	quarantines endpoint using Cisco Tetration not pre-installed, download from https://software.cisco.com/download/home/286259687/type

The Remediation Subsystem

The remediation subsystem consists of the following components:

- the Firepower Management Center's web interface, which you use to set up correlation policies and associate them with remediations, and to track the status of remediation processing

- the remediation API, which enables you to define the data that will be provided to your remediation modules
- the remediation daemon, which passed data to the remediation modules at run time and collects execution status information
- remediation modules, which perform specific responses to correlation policy violations

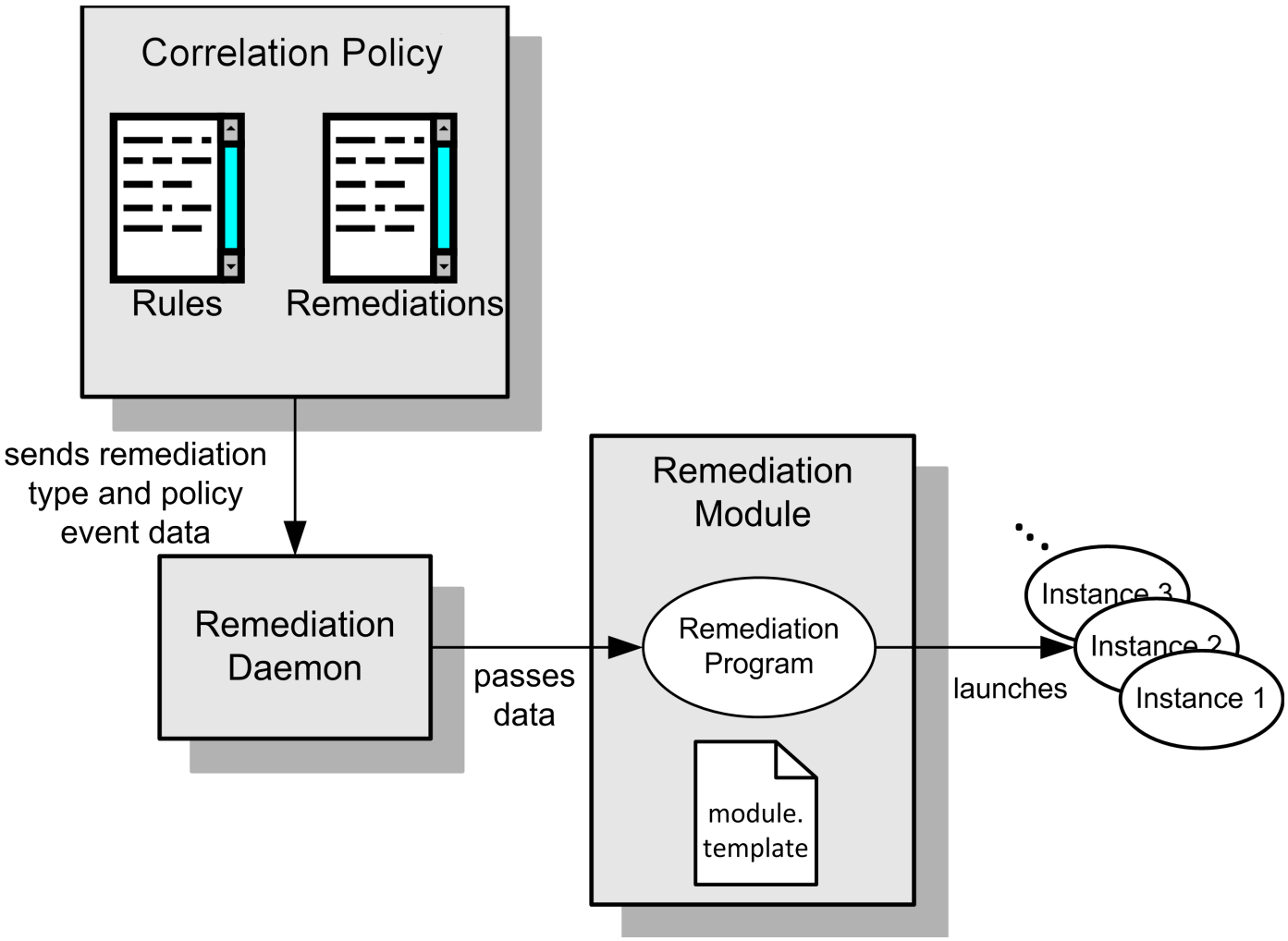
Understanding Remediation Subsystem Architecture

The remediation subsystem has a two-part architecture that is diagrammed in the figure below. The architecture consists of:

- infrastructure components such as the web interface and the remediation daemon which support all remediation modules. The infrastructure components allow you to create and manage all the remediation modules on your Firepower Management Center. The remediation daemon manages the execution of the remediations. See Remediation Subsystem Components, page 1-4 for more details.
- the individual remediation modules which you develop to respond to specific correlation policy violations. See Remediation Module Architecture, page 1-5 for more details.

Remediation Subsystem Components

The following diagram illustrates the main functions of the remediation subsystem and their interactions.



You create remediations in order to respond to rule violations on your network in an automated mode. The Firepower Management Center web interface allows you to define and activate your correlation policies and associate them with remediations. When a policy violation occurs, the remediation subsystem passes the name of the remediation and the event data specified in the `module.template` configuration file to the remediation daemon.

The remediation daemon launches the remediation and passes the correlation event data and instance-specific parameters to your remediation program. It also accepts return codes from the remediation program. The Firepower Management Center uses the return codes for status displays.

The remediation program launches a set of *instances* of the remediation when the associated policy rule triggers. Each instance targets a particular network device. You create instances on the Instance Detail page of the Firepower Management Center web interface. For each instance you provide the necessary instance-specific configuration details such as IP address and password of the target device.

Remediation Module Architecture

Each remediation module that you install on your Firepower Management Center includes one or more remediation types. You assign one or more remediation types to each instance. For information on configuring remediations as responses to policy violations, see the Configuring Remediations chapter in the *Firepower System User Guide*.

Remediation modules include the following components:

- the remediation program, included in the remediation module package at installation. See Planning and Packaging Your Remediation Module, page 2-1.
- a required XML `module.template` file, also included in the remediation module package at installation. This file provides module-level information about your module and its data requirements that the remediation subsystem references each time it launches one of the remediation module's instances. See Communicating with the Remediation Subsystem, page 3-1.
- one XML `instance.conf` file per instance. The Firepower Management Center auto-generates this file each time you configure a new instance of your remediation module.

Using the Remediation Subsystem

You deploy remediations by adding them as responses to specific rules in correlation policies on your Firepower Management Center. You define the associations of correlation policies and remediations using the Firepower Management Center web interface.

To deploy a remediation module, you must:

1. Identify the condition you want to mitigate and the actions that appropriately resolve that condition in your environment. These actions are the main functions your custom remediation program must implement.

   If you can use a Cisco-provided remediation module, skip directly to step 6.Install the module on the Firepower Management Center using the web interface as described in Installing Your Module, page 2-13., page 1-6.
2. If you need to produce a custom remediation module, familiarize yourself with the data elements obtainable from the remediation subsystem. See Data Available from the Remediation Subsystem, page 2-1.
3. If you develop a custom remediation module you must also create a module template file to be included in your module package. See Communicating with the Remediation Subsystem, page 3-1 for the format and syntax of the file.
4. Write your remediation program so that it addresses all the functions necessary for the desired remediations. You can write your remediation module programs in bash, tsch, Perl or C. Develop your program using the technical guidance in Notes for Remediation Program Developers, page 4-3.
5. Package your remediation module as described in Packaging Your Module, page 2-12.
6. Install the module on the Firepower Management Center using the web interface as described in Installing Your Module, page 2-13.
7. Ensure that the individual remediation types in your remediation module are assigned as responses to the correct correlation rules in your active correlation policies. See the *Firepower System User Guide* for procedure details.

Remediation Resources

In addition to this document, other resources you can use to create your remediation modules include:

- a remediation SDK with sample program code in C or Perl that generates syslog alerts and demonstrates how a module can interact with your network. See Working with the Remediation SDK, page 4-1 chapter of this document for detailed information. The SDK can be downloaded from the Support site.

- the `module.template` schema (`module.template.xsd`), which is located on the Firepower Management Center at `/etc/sf/remediation/module.template.xsd`.

The following table describes some of the topics explained in the documentation and where to look for more information.

Table 1-3 Remediation Resources

To learn more about...	See ...
the sample remediation module and the general procedure for creating, installing, and configuring one	Working with the Remediation SDK, page 4-1
writing your remediation program	Planning and Packaging Your Remediation Module, page 2-1
creating the `module.template` file	Communicating with the Remediation Subsystem, page 3-1
packaging your remediation module so you can install it on the Firepower Management Center	Packaging Your Module, page 2-12
installing your remediation module	Installing Your Module, page 2-13
configuring your remediations as responses to security policy violations	the Configuring Remediations chapter in the *Firepower System User Guide*

Planning and Packaging Your Remediation Module

Planning the development of a custom remediation module consists of the tasks listed in the following table, which indicates where to find information and guidance on each task area.

Table 2-1 Remediation Module Planning Tasks

For guidance on ...	Look in...
performing a functional analysis and the importance of understanding the remediation subsystem concept of operations	Overview of the Development and Installation Process, page 4-2
reviewing the data available from the remediation subsystem	Data Available from the Remediation Subsystem, page 2-1
using the return code function of the remediation subsystem	Data Returned by Modules, page 2-12
coordinating your software development and generating the `module.template` file	Communicating with the Remediation Subsystem, page 3-1
packaging the remediation module and installing it	Packaging and Installing Your Module, page 2-12

Data Available from the Remediation Subsystem

Custom remediation modules can receive two kinds of data from the remediation subsystem:

- event data, which includes a variety of data about the correlation policy that was violated and about the original triggering event that caused the policy violation
- instance configuration data, which includes values entered in the web interface when an instance of a remediation is configured

These two types of data incorporate both the data about the network traffic or change that triggered the rules in the violated policy, and the configured instance of the remediation that runs in response to that policy violation. See “Correlation Policies” and “Configuring Remediations” in the *Firepower System User Guide* for more information about creating, configuring and using correlation policies and remediations.

See the following sections for more information:

- Event Data, page 2-2 describes how event data is provided to your remediation module and lists the correlation event data available to your module.
- Instance Configuration Data, page 2-8 explains how `instance.config` files are made available to your remediation module and describes the types of data they may include.

Event Data

Event data is one type of information available to your remediation module. Event is information about intrusion, correlation, and other event types that the Firepower Management Center generates when rules in a correlation policy trigger. You specify the event data fields to be sent for each remediation type in your module using the `pe_item` element in the `module.template` file.

When the remediation daemon sends event data to your remediation module, it passes the name of the remediation first, followed by the `pe_item` fields in the order in which they appear in `module.template`.

The remediation daemon handles any undefined `pe_item` fields from the database differently depending on whether they field is marked as optional or required in module.template. See Handling Undefined Data Elements, page 4-6.

For details on specifying event data for remediations, see Defining Remediation Types, page 3-20. When specifying the `pe_item` element, you must use the field names provided in the tables below.

The following table describes data available about the original event that triggered the correlation policy violation. Note that some fields in this table are event specific. These fields are set to zero when not applicable for the specific type of triggering event.

Table 2 Triggering Event Data

Name	Description	Field	Type	Bytes
Transport Protocol	The transport protocol (TCP, UDP, IP, ICMP) of the packet that triggered the intrusion or discovery event that caused the policy violation.	ip_protocol	uint8_t	1
Network Protocol	The network protocol (for example, ethernet) of the packet that triggered the intrusion or discovery event that caused the policy violation.	net_protocol	uint16_t	2
Triggering Event Type	A numeric identifier for the type of event that triggered the correlation event. Values are: 1 = intrusion 2 = network discovery, connection, or connection summary 3 = user awareness 4 = white list	event_type	uint8_t	1
Triggering Event ID	An internal identifier for the event that triggered the correlation event. Set only for intrusion events. Set to 0 for other event types.	event_id	uint32_t	4
Triggering Event Time	Content varies by event type: for intrusion, network discovery, connection, and user awareness events: UNIX timestamp of the triggering event for connection summaries: correlation event time (that is, policy_tv_sec) for white list events: set to 0	tv_sec	uint32_t	4
Triggering Event Time (usec)	The microsecond increment of the event time. Set to 0 if granularity is not available.	tv_usec	uint32_t	4
Triggering Event Description	A text description of the original event that triggered the correlation event. Content varies by event type.	description	char *	Max 1024

Table 2 Triggering Event Data (continued)

Name	Description	Field	Type	Bytes
Triggering Event Sensor ID	The internal identifier of the sensor where the triggering event occurred. Primarily for Cisco internal use, not typically used for remediations.	sensor_id	uint32_t	4
Triggering Event Generator ID	Content varies by event type: for intrusion events: the generator ID (GID) for the event. See the *Firepower System User Guide* for a complete list of GIDs. for network discovery and connection events: the network discovery event type. for connection summaries: set to 4 for all. for user awareness events: the user awareness event type. for white list events: set to 0. Primarily for Cisco internal use and not typically used for remediations.	sig_gen	uint32_t	4

Table 2 Triggering Event Data (continued)

Name	Description	Field	Type	Bytes
Triggering Event Signature ID	Content varies by event type: for intrusion events: the signature ID (SID) for the event. May not match the SID displayed in the user interface. for network discovery and connection events: network discovery event subtype. for connection summaries: set to 17 for all. for user awareness events: user awareness event subtype for white list events: set to 0. Primarily for Cisco internal use and not typically used for remediations.	sig_id	uint32_t	4

Table 2 Triggering Event Data (continued)

Name	Description	Field	Type	Bytes
Impact Flags	Impact flag value of the event. The low-order eight bits indicate the impact level. Values are: 0x01 (bit 0) - Source or destination host is in a network monitored by the system. 0x02 (bit 1) - Source or destination host exists in the network map. 0x04 (bit 2) - Source or destination host is running a server on the port in the event (if TCP or UDP) or uses the IP protocol. 0x08 (bit 3) - There is a vulnerability mapped to the operating system of the source or destination host in the event. 0x10 (bit 4) - There is a vulnerability mapped to the server detected in the event. 0x20 (bit 5) - The event caused the managed device to drop the session (used only when the device is running in inline, switched, or routed deployment). Corresponds to blocked status in the Firepower System web interface. 0x40 (bit 6) - The rule that generated this event contains rule metadata setting the impact flag to red. The source or destination host is potentially compromised by a virus, trojan, or other piece of malicious software. 0x80 (bit 7) - There is a vulnerability mapped to the client detected in the event. (version 5.0+ only) The following impact level values map to specific priorities on the Firepower Management Center. An `x` indicates the value can be 0 or 1: gray (0, unknown): `00x00000` red (1, vulnerable): `xxxx1xxx`, `xxx1xxxx`, `x1xxxxxx`, `1xxxxxxx` (version 5.0+ only) orange (2, potentially vulnerable): `00x0011x` yellow (3, currently not vulnerable): `00x0001x` blue (4, unknown target): `00x00001`	impact_flags	uint32_t	4

The following table describes the data available about each correlation event. Note that some of the data elements are not populated for certain event types.

Table 3 Correlation Event Data

Name	Description	Field	Type	Bytes
Correlation Event Time	UNIX timestamp of when the correlation event was generated.	policy_tv_sec	uint32_t	4
Correlation Event ID	The internal identification number of the event generated by the sensor. Set only for intrusion events. Primarily for Cisco internal use and not typically used for remediations.	policy_event_id	uint32_t	4
Correlation Appliance ID	The internal identification number of the Firepower Management Center that generated the correlation event. Primarily for Cisco internal use and not typically used for remediations.	policy_sensor_id	uint32_t	4
Correlation Policy ID	The internal identification number of the of the correlation policy that was violated by the triggering event. Primarily for Cisco internal use and not typically used for remediations.	policy_id	uint32_t	4
Correlation Rule ID	The internal identification number of the correlation rule that triggered the correlation event. Primarily for Cisco internal use and not typically used for remediations.	rule_id	uint32_t	4
Correlation Rule Priority	The priority assigned to the rule for the correlation policy that generated the event. The rule may have a different priority in another policy. Value: 0 – 5 (0 = no priority)	priority	uint32_t	4
Event-Defined Mask	A bit field in the correlation event message that indicates which of the fields that follow the mask are valid. See Table 2-4Event Defined Values, page 2-6 for the values. Primarily for Cisco internal use and not typically used for remediations.	defined_mask	uint32_t	4

The following table defines the mask values for the correlation event message fields. These values are used in the correlation event message to indicate which of the fields that follow the mask are valid.

Table 2-4 Event Defined Values

Correlation Event Field	Mask Value
Event Impact Flags	0x00000001
IP Protocol	0x00000002
Network Protocol	0x00000004
Source IP	0x00000008
Source Host Type	0x00000010

Table 2-4 Event Defined Values (continued)

Correlation Event Field	Mask Value
Source VLAN ID	0x00000020
Source Fingerprint ID	0x00000040
Source Criticality	0x00000080
Source Port	0x00000100
Source Server	0x00000200
Destination IP	0x00000400
Destination Host Type	0x00000800
Destination VLAN ID	0x00001000
Destination Fingerprint ID	0x00002000
Destination Criticality	0x00004000
Destination Port	0x00008000
Destination Server	0x00010000
Source User	0x00020000
Destination User	0x00040000

The following table describes the data available about the source host involved in the intrusion event, or the only host involved in any other discovery event that caused the correlation policy violation. Note that only the source IP address is guaranteed to be populated.

Table 5 Source Host Data

Name	Description	Field	Type	Bytes
IP Address	The IP address of the source host in the event that triggered the policy violation. For a discovery event, the host or initiator host's IP address.	src_ip_addr	uint32_t	4
Host Type ID	The host's recognized type (for example, router, bridge); discovery events only.	src_host_type	uint8_t	1
VLAN ID	The host's VLAN ID; discovery events only.	scr_vlan_id	uint16_t	2
OS Vendor	The vendor of the host's identified operating system; discovery events only.	src_os_vendor	char*	max 255
OS Product	The host's identified operating system; discovery events only.	src_os_product	char*	max 255
OS Version	The version number of the host's identified operating system; discovery events only.	src_os_version	char*	max 255
Host Criticality	A user-defined value in host and connection events.	src_criticality	uint16_t	2

The following table describes the data available about the source host's server, or only server identified in the event that caused the correlation event. Note that only the transport protocol is guaranteed to be populated

Table 6 Source Server Data

Name	Description	Field	Type	Bytes
Port	Port on which the identified server is running. For intrusion events, port is populated only if the protocol is TCP or UDP.	src_port	uint16_t	2
Server	Server (for example, HTTP, SMTP) identified in the event that caused the policy violation.	src_service	char	max 255

The following table describes the data available about the destination host. This data is only available for intrusion events.

Table 7 Destination Host Data

Name	Description	Field	Type	Bytes
IP Address	The IP address of the destination host in the event that triggered the policy violation.	dest_ip_addr	uint32_t	4
Host Type ID	The destination host's recognized type (for example, router, bridge).	dest_host_type	uint8_t	1
VLAN ID	The destination host's VLAN ID.	dest_vlan_id	uint16_t	2
OS Vendor	The vendor of the host's identified operating system; discovery events only.	dest_os_vendor	char*	max 255
OS Product	The host's identified operating system; discovery events only.	dest_os_product	char*	max 255
OS Version	The version number of the host's identified operating system; discovery events only.	dest_os_version	char*	max 255
Host Criticality	A user-defined value in; discovery host and connection events.	dest_criticality	uint16_t	2

The following table describes the data available about the destination host's server, or the only server identified in the event that caused the correlation event. Note that only the transport protocol is guaranteed to be populated.

Table 8 Destination Server Data

Name	Description	Field	Type	Bytes
Destination Port	Port on which the identified server is running. In the case of intrusion events, the port is populated only if the protocol is identified as TCP or UDP.	dest_port	uint16_t	2
Destination Server	Server (for example, HTTP, SMTP) identified in the event that caused the policy violation.	dest_service	char	max 255

Instance Configuration Data

When a user configures a new instance of your module, they provide data requested in your `module.template` document. The values provided by the user are then written into the `instance.conf` document for use by your remediation program.

For each configured instance of a remediation, the remediation subsystem places an `instance.conf` document in a directory with the same name as the instance. This directory is created in the directory where your module was uploaded and installed. For example, if your module is called Firewall, it is

uploaded into a directory called `firewall`. If you then configure an instance called `block_tokyo`, the remediation subsystem creates a directory called `block_tokyo` in your `firewall` directory and places the `instance.conf` there. The directory path appears as follows:

```
/var/sf/remediation/firewall/block_tokyo/instance.config
```

See Packaging Your Module, page 2-12 for more information on the directories where your module files reside.

Your module must be able to open, read, parse, and close the `instance.conf` file.

Each `instance.conf` document contains a top level element called `instance`. The `instance` element has two child elements: `config` and `remediation`. The following table describes the attributes and elements available to the instance element.

Table 2-9 instance Attributes and Child Elements

Name	Type	Description
name	attribute	Ties the data in the document to the named, configured instance and reflects the name of the instance specified by the configuring user.
config	element	Contains the data entered into the instance configuration fields on the web interface at configuration.
remediation	element	Contains the data entered into the web interface when configuring the remediation for an instance.

For more information about the data provided in the `config` and `remediation` elements, see the following:

- The config Element, page 2-9
- The remediation Element, page 2-11

The config Element

The `config` element contains the data entered into the fields rendered on the web interface in response to the `config_template` element in that remediation module's `module.template` document. These fields are translated back into the elements used to specify them in the `module.template` document, and further specified using the name provided as an attribute of the element rather than a child element. They can include the following types of fields:

- boolean
- string
- integer
- password
- host
- netmask
- network
- ipaddress
- enumeration
- list

See Defining the Configuration Template, page 3-4 for more details on how these fields are specified in the `module.template` file.

For example, if the `module.template` document contains the following `config_template` element definition:

```
<config_template>
<ipaddress>
    <name>host_ip</name>
    <display_name>Host IP</display_name>
</ipaddress>
<string>
    <name>user_name</name>
    <display_name>Username</display_name>
    <constraints>
          <pcre>\S+</pcre>
    </constraints>
</string>
<password>
    <name>login_password</name>
    <display_name>Login Password</display_name>
</password>
</config_template>
```

The Instance Configuration screen for that element contains the following three fields:

- Host IP, which takes an IP address value.
- Username, which takes a string value that may not contain white space characters.
- Login Password, which takes a string value identified as a password.

Suppose a user configures an instance, named AdminInstance, of the remediation module and provides the following values:

Table 2-10 Sample Values

Field	Value
Host IP	192.1.1.1
Username	adminuser
Login Password	3admin3

The `instance.conf` will contain the following:

```
<instance name="AdminInstance">
<config>
    <ipaddress name="host_ip">192.1.1.1</ipaddress>
    <string name="user_name">adminuser</string>
    <password name="login_password">3admin3</password>
</config>
```

Note that the above example does not include `</instance>`. This is because the `instance.conf` document for this example instance would go on to include the `remediation` element discussed next in this section. If you do not require additional remediation configuration in your module, the `instance.conf` returned for that module does **not** include remediation elements.

The remediation Element

The `instance` element contains a `remediation` element for each remediation configured for that instance. Each `remediation` element has, as an attribute, the name of the remediation instance (entered into the web interface at the time the instance is configured) and the type of the remediation, which was initially provided by the `remediation_type` element in the `module.template` document. For more information about the `module.template` file, see Communicating with the Remediation Subsystem, page 3-1.

In addition, `remediation` elements can contain `config` elements. These function in the same way as `config` elements that are child elements of `instance`, but use data originally specified in the `config_template` element that is a child of `remediation_type` in the `module.template` document. The following describes these attributes and elements.

Table 2-11 remediation Attributes and Child Elements

Name	Type	Description
name	attribute	Ties the data in the document to the named, configured remediation and reflects the name specified by the configuring user.
type	attribute	Provides the type of remediation configured in this instance.
config	element	Contains the data entered into the remediation configuration fields on the web interface at configuration.

For example, suppose the `module.template` document in the example provided in The config Element, page 2-9 continues with the following:

```
<remediation_type name="acl_insert">
<display_name>ACL Insertion</display_name>
<policy_event_data>
    <pe_item>src_ip_addr</pe_item>
    <pe_item>src_port</pe_item>
    <pe_item>src_protocol</pe_item>
    <pe_item>dest_ip_addr</pe_item>
    <pe_item>dest_port</pe_item>
    <pe_item>dest_protocol</pe_item>
</policy_event_data>
<config_template>
    <integer>
        <name>acl_num</name>
        <display_name>ACL Number</display_name>
    </integer>
</config_template>
</remediation_type>
```

The Instance Detail page that allows you to add remediations to a created instance contains the remediation type “ACL Insertion”. Adding “ACL Insertion” to the instance takes the user to a page that includes a name field, which populates the name attribute value for that remediation element in the `instance.conf`, and a field labelled ACL Number, which accepts an integer value.

Suppose a user adds this remediation to the AdminInstance instance and provides the following values:

Table 2-12 Sample Values

Field	Value
Remediation Name	AdminRemediation
ACL Number	55

The `instance.conf` document written when the user saved the example configuration values would, after the section provided in the example in The config Element, page 2-9, continue as follows:

```
<remediation name="AdminRemediation" type="acl_insert">
<config>
    <integer="acl_num">55</integer>
</config>
</remediation>
```

Note that if no more remediations were added to the instance, the `instance.conf` should be terminated with `</instance>` at this point.

Data Returned by Modules

Remediation modules must return exit status codes, known as return codes, to the Firepower Management Center. The Table View of Remediations in the Firepower Management Center web interface displays a result message for each remediation launched. The return code from the remediation program determines the result message displayed.

Return codes must be integers in the 0 to 255 range inclusive, as defined in the following table.

Table 2-13 Return Code Ranges

Range	Use
0 - 128	Reserved for Cisco predefined return codes
129 - 255	Available for custom remediations

See Defining Exit Statuses, page 3-22 for the list of predefined codes and for directions on creating custom codes.

Packaging and Installing Your Module

The remediation API requires that you package your remediation modules. The files that make up your module must be provided in a gzipped tar file.

See the following sections for more information:

- Packaging Your Module, page 2-12 provides helpful tips for packaging your binaries and `module.template` files for upload and installation.
- Installing Your Module, page 2-13 explains how to install your remediation module on the Firepower Management Center.

Packaging Your Module

When packaging your remediation files for installation, keep in mind the following:

- Remediation modules must be packaged in a gzipped tarball (`.tar.gz` or `.tgz`) before you install them.
- When you install the module, the package is extracted into `/var/sf/remediation/remediation_directory` where `remediation_directory` is a combination of the `name` attribute of the module's `module` element and the data in the `version` element.

  For example, one of the default remediation modules shipped with the Firepower Management Center is the Cisco PIX Shun module. That module resides in `/var/sf/remediation/cisco_pix_1.0`.
- When extracted, your remediation module's `module.template` document must reside in the top level of the directory created to contain that module package.
- As instances of remediations are created, they are saved in a directory created in your module directory and named for the instance.

  For example, instances of the Cisco PIX Shun module might reside in `/var/sf/remediation/cisco_pix_1.0/PIX_01` and `/var/sf/remediation/cisco_pix_1.0/PIX_02`.

For example, you upload and install a module that is packaged in firewall.tgz and is named in the `module.template` as `firewall` with a version value of `1.0`. The system installs the module in the following directory: `/var/sf/remediation/firewall_1.0`. That directory contains your module.template file and your program binary. When you add an instance to the remediation module and name it block_tokyo, the system creates the following directory:

`/var/sf/remediation/firewall_1.0/block_tokyo`

and places the `instance.conf` file for `block_tokyo` in it.

Installing Your Module

Once you have correctly packaged your remediation module, use the Modules page to install it.

To install a new module on the Remediation API:

1. Select **Policies > Actions > Modules**.

   The Installed Remediation Modules page appears.
2. Click **Browse** to navigate to the location where you saved the tar.gz file that contains the custom remediation module.
3. Click **Install**.

   The custom remediation module installs.
4. Select **Policies > Actions > Modules**.

   The Installed Remediation Modules table lists the module just installed. The Module Name, Version, and Description columns match the information defined in the `module.template` file.
5. Add instances of your new module and associate remediations to each instance, as described in the *Firepower System User Guide*.

You can use the Modules page to view the remediation modules installed on the Firepower Management Center. The list displays custom remediation modules and Cisco-provided ones. You can also delete your custom modules.

To view or delete a module from the Remediation API:

1. Select **Policies > Actions > Modules**.

The Installed Remediation Modules page appears.

2. Perform one of the following actions:

- Click the View icon to view the module.

  The Module Detail page appears.

- Click the Delete icon next to the module you want to delete. You **cannot** delete default modules provided by Cisco.

  The remediation module is deleted from the Remediation API.

Communicating with the Remediation Subsystem

Your remediation module must receive information from the Firepower Management Center remediation subsystem to successfully perform its function. You configure the information that your module receives in an XML file called `module.template`. Without it, the remediation subsystem cannot interact with your remediation module.

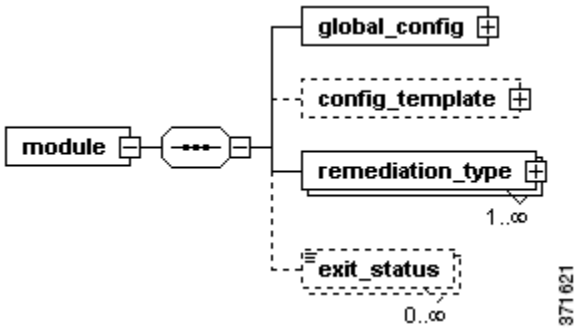
The `module.template` XML file allows you to specify:

- a set of module-level declarations such as the name and version of your remediation module, a short descriptive text, and the name of the binary file for your remediation program
- the information the module requires from the user when the user configures remediation instances in the Firepower Management Center user interface
- the specific remediation actions, known as remediation types, that the module can perform and the correlation event data each remediation type requires
- any custom return codes and exit status messages that your remediation program returns to the Firepower Management Center

Before writing a `module.template` for your remediation module, you should understand the `module.template` schema (`module.template.xsd`). The schema defines the elements (or tags used to contain data) and attributes (or data used to modify the data contained in an element) you can use to provide information to the remediation subsystem. The `module.template` schema is located on the DC at `/etc/sf/remediation/module.template.vsd`.

The top-level element in `module.template` is `module`, in which you specify the name of the remediation module using the `name` attribute. The `name` attribute is required and accepts a string value between 1 and 64 alphabetic characters.

Caution: You cannot use white space in the module's `name` attribute value. In addition, you cannot use punctuation marks except for underscore (_) or dash (-).



Some XML editors can read the `module.template` schema and automatically generate a `module.template` file with a namespace and schema declaration, with the top level element and child elements and attributes. If you choose not to use such an editor, you must include the child elements manually.

Caution: If you set your XML editor to auto-generate the namespace and schema location, you must delete those lines before including the final version of `module.template` in your installation package.

The following example illustrates the `module` element with only the `name` attribute defined.

```
<module name="example_module">
<global_config>
   <display_name/>
   <version/>
   <binary/>
</global_config>
<remediation_type name="">
   <display_name/>
</remediation_type>
</module>
```

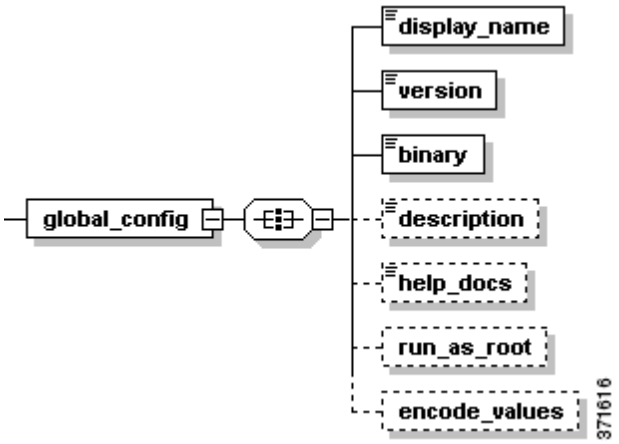
See the following sections for details about writing the rest of `module.template`:

- Defining the Global Configuration, page 3-2 explains how to use the `global_config` element to define the name that appears for your module on the Modules page, as well as the module's version, binary location, and its description.
- Defining the Configuration Template, page 3-4 explains how to use the `config_template` element to define the configuration information that your module requires the user to specify from the web interface.
- Defining the Global Configuration, page 3-2 explains how to use the `remediation_type` element to define the remediations the module can launch and the correlation event data that each remediation requires.
- Defining Exit Statuses, page 3-22 explains how to use the `exit_status` element define the custom exit statuses your module returns to the remediation subsystem.

Defining the Global Configuration

The first required section of `module.template` uses the `global_config` element to define global configuration information. These attributes include the module's name and description, which appear in the list of remediation modules displayed on the Modules page of the Firepower Management Center user interface. The global information also includes the module's version and the location of the executable program that runs when a remediation is triggered.

The following portion of the `module.template` schema diagram illustrates the child elements of the `global_config` element.



The following table describes the child elements available to the `global_config` element.

Table 3-1 global_config Child Elements

Name	Description	Required?
`display_name`	Specifies the name that appears for this remediation module on the Modules page. The display name can contain only alphanumeric characters and white spaces and must be between 1 and 127 characters long. It must be unique across remediation modules.	yes
`version`	Specifies the version of the remediation module. This value appears on the Modules page. The value for the version element must begin and end with numeric characters, but may contain period (.) characters. Note: The combination of the `name` attribute of the `module` element and the data in the `version` element must be unique across remediation modules.	yes
`binary`	Specifies the UNIX filename of the binary that makes up your remediation module.	yes
`description`	Provides a description of the remediation module and its available remediations. The `description` element appears on the Modules page. Descriptions with more than 255 characters are truncated.	yes
`run_as_root`	Sets a flag that allows the remediation module to run as root on the Cisco appliance where it is installed. Caution: Cisco recommends that you use this element only if absolutely necessary.	no
`encode_values`	Sets a flag that HTML-encodes user input. This allows users to enter input that might otherwise be unintentionally interpreted by the XML processor. Note: If you use this element, your remediation module **must** handle HTML decoding as part of its input handling.	no

Consider the following XML code, which illustrates the global configuration portion of a `module.template` file.

```
<global_config>
<display_name>My Firewall</display_name>
<binary>firewall_block.pl</binary>
<description>Dynamically apply firewall rules to my firewall.</description>
<version>1.0</version>
<run_as_root/>
</global_config>
```

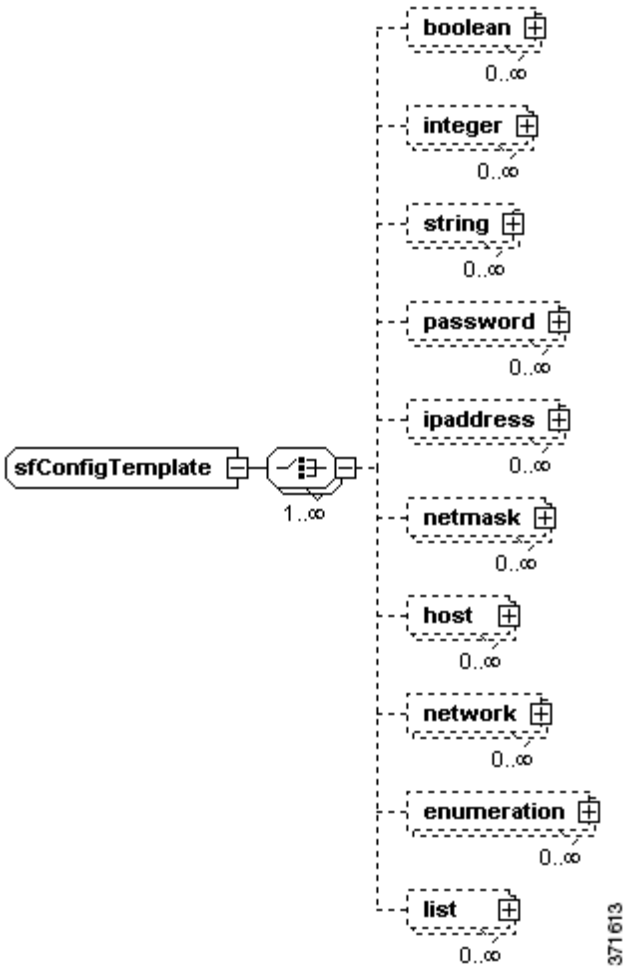
In this example, the remediation module is represented by the name My Firewall in the web interface. It runs version 1.0 of a program called `firewall_block.pl`, which you install using the Firepower Management Center (see Packaging and Installing Your Module, page 2-12 for more information). The program dynamically applies firewall rules to a specific firewall and runs as root on the Firepower Management Center.

Defining the Configuration Template

The `config_template` child element of the `module` element specifies the types of information the user must provide when configuring the instances that this remediation module executes (see Instance Configuration Data, page 2-8). The user provides the information specified in this element via the Firepower Management Center user interface. Each `module` element may contain only one direct child `config_template` element and this element applies to all instances that are configured.

Note, however, that each `remediation_type` element in `module.template` can also contain a child `config_template` element. The `config_template` child element under `remediation_type` allows you to define information that the user must provide for each of the different remediation types. So a user will have to configure general instance-level fields using the `config_template` element in the `module` portion, and then, optionally, an additional set of `config_template` fields specific to the remediation type being executed by the instance. For more information, see Defining Remediation Types, page 3-20.

The following diagram illustrates the child elements available to the `config_template` element.



The `config_template` element allows you to render several basic field types in the web interface. You choose which `config_template` child elements to use depending on the data you need to collect from the user for the remediation module. All child elements of `config_template` are optional and can be used as many times as needed within a `config_template` element. Fields are rendered on the web interface in the order in which they are included in the `config_template` element.

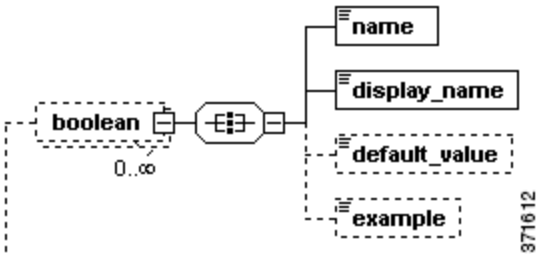
See the following sections for more information on the child elements that represent the fields you can use to collect configuration information on the instance configuration and remediation configuration pages in the web interface:

- The boolean Element, page 3-6
- The integer Element, page 3-7
- The string Element, page 3-8
- The password Element, page 3-9
- The ipaddress Element, page 3-11
- The netmask Element, page 3-12
- The host Element, page 3-13
- The network Element, page 3-14

The boolean Element

Each `boolean` element you use in a `config_template` represents a true/false choice, which appears as a set of radio buttons labeled **On** or **Off**, users can make in the web interface. If you set the element's `required` attribute to false, an additional radio button is available, labeled **Not Selected**.

The following portion of the `module.template` schema diagram illustrates the `boolean` element's child elements.



When configuring child elements for an occurrence of a `boolean` element, you may only use each available child element once. The following table describes the child elements available to the `boolean` element.

Table 2 boolean Attributes and Child Elements

Name	Type	Description	Required?
required	attribute	Indicates whether specifying a value in the field is optional. This attribute defaults to `true`. You are not required to use this attribute. Therefore, if you do not use it (or if you explicitly set its value to `true`), users must select either **On** or **Off**. If you set the value of the attribute to `false`, the web interface indicates that the choice is optional.	no
`name`	element	Provides context to the remediation module for the value entered in the field. Names may not contain white space and may only contain alphanumeric characters and the underscore (_) and dash (-) character. Names should be unique within a module.	yes
`display_name`	element	Specifies the web interface label for this field.	yes
`default_value`	element	Specifies the default value for this field. If the web interface user does not specify a value, the remediation program uses this value by default.	no

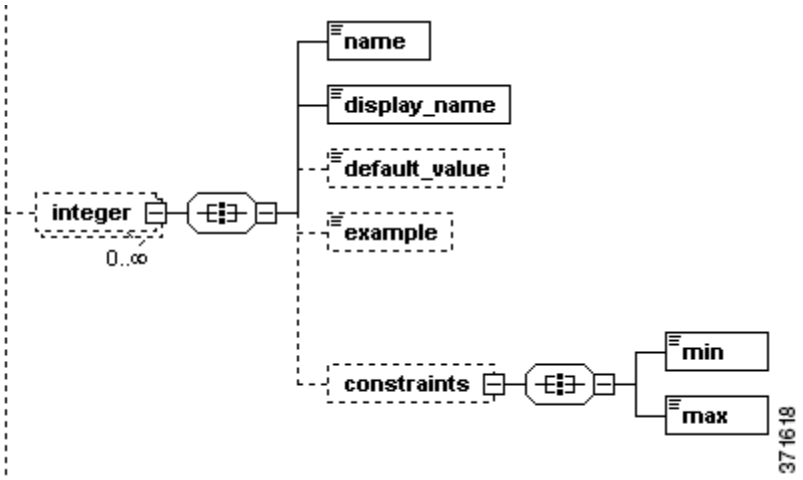
The following portion of a `config_template` element definition indicates that the web interface displays a field labeled "Enabled?" that provides user with two choices: **On** or **Off**. The choice defaults to `true`, that is, the radio button labeled **On** is preselected.

```
<boolean>
<name>process_enabled</name>
<display_name>Enabled?</display_name>
<default_value>true</default_value>
</boolean>
```

The integer Element

Each `integer` element you use in a `config_template` represents a field in the web interface that accepts an integer value.

The following diagram illustrates the child and grandchild elements of the `integer` element.



The following table describes the child elements available to the `integer` element.

Table 3 integer Attributes, Child Elements, and Grandchild Elements

Name	Type	Description	Required?
required	attribute	Indicates whether users must provide a value in the field. This attribute defaults to `true`. You are not required to use this attribute. Therefore, if you do not use it (or if you explicitly set its value to `true`), users must provide a value. If you set the value of the attribute to `false`, the web interface indicates that providing a value is optional.	no
`name`	element	Provides context to the remediation module for the value entered in the field. Names may not contain white space and may only contain alphanumeric characters and the underscore (_) and dash (-) character.Names should be unique within a module.	yes
`display_name`	element	Specifies the web interface label for this field.	yes
`default_value`	element	Specifies the default value for this field. If the web interface user does not specify a value, the remediation program uses this value by default.	no
`example`	element	Provides an example of the input that the remediation module expects to receive. Note: This value is not displayed in the web interface.	no
`constraints`	element	Constrains the values that the user can enter in this field to fall between specified minimum and maximum values, inclusive. The `constraints` element has two child elements: `min` and `max`. Each is an optional, single-occurrence child element that accepts an integer value.	no

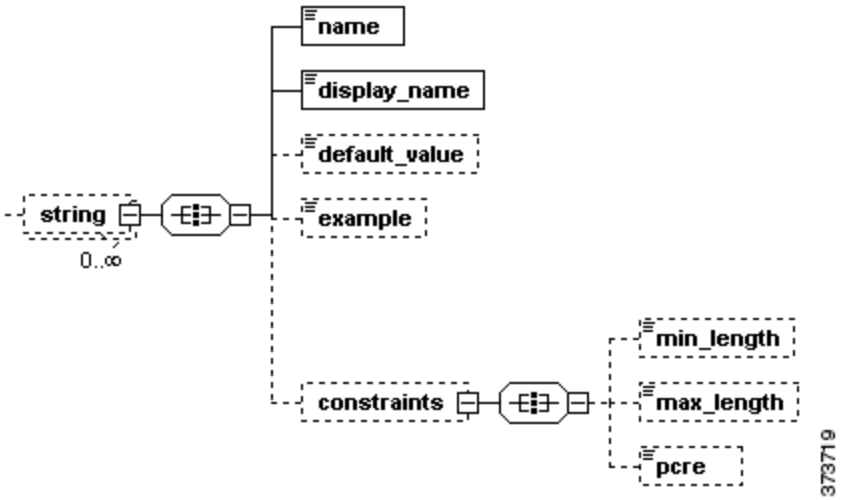
The following portion of a `config_template` element definition indicates that the web interface displays a field labeled “Rate”, which accepts an integer value between 0 and 500 but defaults to 430.

```
<integer>
<name>rate</name>
<display_name>Rate</display_name>
<default_value>430</default_value>
<constraints>
    <min>0</min>
    <max>500</max>
</constraints>
</integer>
```

The string Element

Each `string` element you use in a `config_template` represents a field in the web interface that accepts a string value.

The following diagram illustrates the child elements of the `string` element instance.



The following table describes child elements available to the `string` element.

Table 4 string Attributes, Child Elements, and Grandchild Elements

Name	Type	Description	Required?
required	attribute	Indicates whether users must provide a value in the field. This attribute defaults to `true`. You are not required to use this attribute. Therefore, if you do not use it (or if you explicitly set its value to `true`), users must provide a value. If you set the value of the attribute to `false`, the web interface indicates that providing a value is optional.	no
`name`	element	Provides context to the remediation module for the value entered in the field. Names may not contain white space and may only contain alphanumeric characters and the underscore (_) and dash (-) character.Names should be unique within a module.	yes
`display_name`	element	Specifies the web interface label for this field.	yes

Table 4 string Attributes, Child Elements, and Grandchild Elements (continued)

Name	Type	Description	Required?
`default_value`	element	Specifies the default value for this field.If the web interface user does not specify a value, the remediation program uses this value by default.	no
`example`	element	Provides an example of the input that the remediation module expects to receive. Note: This value is not displayed in the web interface.	no
`constraints`	element	Constrains the values that the user can enter in this field. The `constraints` element has three child elements: `min_length`, `max_length` and `pcre`. The `min_length` and `max_length` elements are optional, single-occurrence child elements that accept integer values and specify a range for the acceptable length of string values. The `pcre` element is optional; use it to specify a Perl-compatible regular expression that provides additional constraints.	no

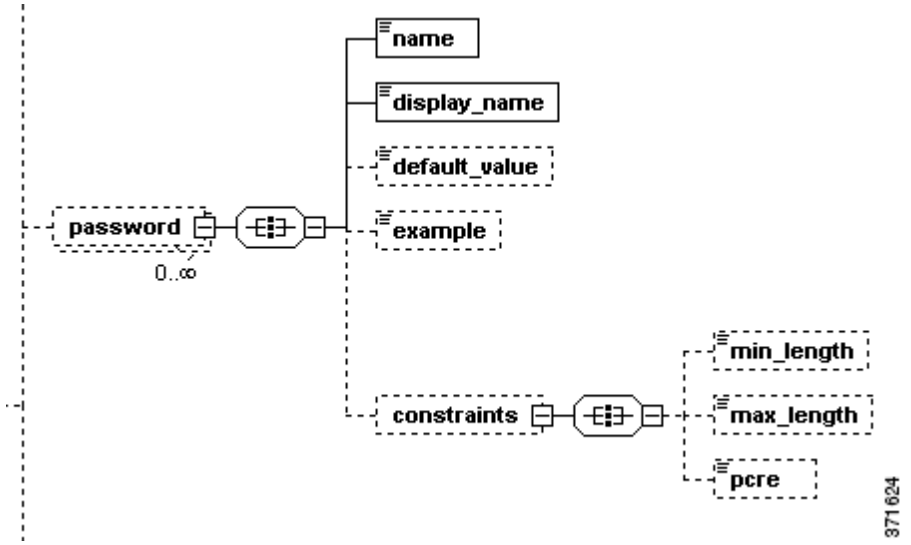
The following portion of a `config_template` element definition indicates that the web interface displays a field labeled “Username”, which accepts a string value that is at least eight characters long and does not use white spaces.

```
<string>
    <name>user_name</name>
    <display_name>Username</display_name>
    <constraints>
        <min_length>8</min_length>
        <pcre>\S+</pcre>
    </constraints
</string>
```

The password Element

Each `password` element you use in a `config_template` represents a field in the web interface that accepts a string comprised of alphanumeric characters.

The following diagram illustrates the child and grandchild elements of the `password` element instance.



The following table describes the child elements available to the `password` element.

Table 5 password Attributes, Child Elements, and Grandchild Elements

Name	Type	Description	Required?
required	attribute	Indicates whether users must provide a value in the field. This attribute defaults to `true`. You are not required to use this attribute. Therefore, if you do not use it (or if you explicitly set its value to `true`), users must provide a value. If you set the value of the attribute to `false`, the web interface indicates that providing a value is optional.	no
`name`	element	Provides context to the remediation module for the value entered in the field. Names may not contain white space and may only contain alphanumeric characters and the underscore (_) and dash (-) character. Names should be unique within modules.	yes
`display_name`	element	Specifies the web interface label for this field.	yes
`default_value`	element	Specifies the default value for this field. If the web interface user does not specify a value, the remediation program uses this value by default.	no
`example`	element	Provides an example of the input that the remediation module expects to receive. Note: This value is not displayed in the web interface.	no
`constraints`	element	Constrains the values that the user can enter in this field. The `constraints` element has three child elements: `min_length`, `max_length` and `pcre`. The `min_length` and `max_length` elements are optional, single-occurrence child elements that accept integer values and specify a range for the acceptable length of password values. The `pcre` element is optional; use it to specify a Perl-compatible regular expression that provides additional constraints.	no

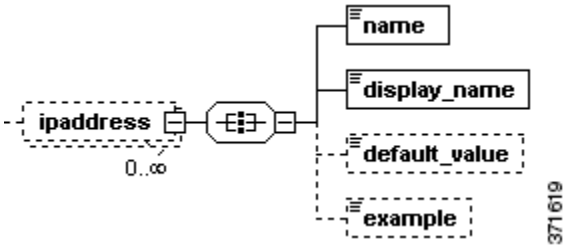
The following portion of a `config_template` element definition indicates that the web interface displays a field labeled “Login Password”, which accepts an alphanumeric string between 6 and 12 characters long.

```
<password>
    <name>login_password</name>
    <display_name>Login Password</display_name>
    <constraints>
        <min_length>6</min_length>
        <max_length>12</max_length>
    </constraints>
</password>
```

The ipaddress Element

Each `ipaddress` element you use in a `config_template` represents a field in the web interface that accepts a single IP address. IP addresses may be entered in the form of a fully formed dotted quad (for example, 1.1.1.1).

The following diagram illustrates the child elements of the `ipaddress` element.



When configuring child elements for an occurrence of an `ipaddress` element, you may only use each available child element once. The following table describes the child elements available to the `ipaddress` element.

Table 6 ipaddress Attributes and Child Elements

Name	Type	Description	Required?
required	attribute	Indicates whether users must provide a value in the field. This attribute defaults to `true`. You are not required to use this attribute. Therefore, if you do not use it (or if you explicitly set its value to `true`), users must provide a value. If you set the value of the attribute to `false`, the web interface indicates that providing a value is optional.	no
`name`	element	Provides context to the remediation module for the value entered in the field. Names may not contain white space and may only contain alphanumeric characters and the underscore (_) and dash (-) character. Names should be unique within modules.	yes
`display_name`	element	Specifies the web interface label for this field.	yes
`default_value`	element	Specifies the default value for this field.	no
`example`	element	Provides an example of the input that the remediation module expects to receive. Note: This value is not displayed in the web interface.	no

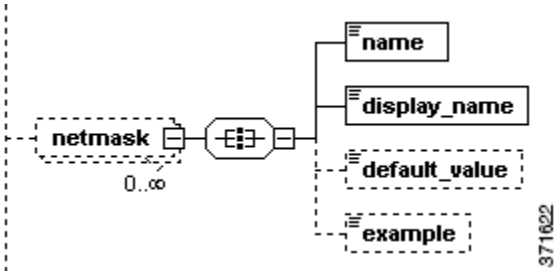
The following portion of a `config_template` element definition indicates that the web interface displays a field labeled “Mail Server,” which accepts a single IP address.

```
<ipaddress>
<name>mail_server</name>
<display_name>Mail Server</display_name>
</ipaddress>
```

The netmask Element

Each `netmask` element you use in a `config_template` represents a field in the web interface that accepts netmask values. Netmask values can be denoted by a dotted quad (255.255.255.255) or a CIDR mask (/8).

The diagram illustrates the child elements of the `netmask` element.



When configuring child elements for an occurrence of a `netmask` element, you may only use each available child element once. The following table describes the child elements available to the `netmask` element.

Table 7 netmask Attributes and Child Elements

Name	Type	Description	Required?
required	attribute	Indicates whether users must provide a value in the field. This attribute defaults to `true`. You are not required to use this attribute. Therefore, if you do not use it (or if you explicitly set its value to `true`), users must provide a value. If you set the value of the attribute to `false`, the web interface indicates that providing a value is optional.	no
`name`	element	Provides context to the remediation module for the value entered in the field. Names may not contain white space and may only contain alphanumeric characters and the underscore (_) and dash (-) character. Names should be unique within modules.	yes
`display_name`	element	Specifies the web interface label for this field.	yes
`default_value`	element	Specifies the default value for this field. If the web interface user does not specify a value, the remediation program uses this value by default.	no
`example`	element	Provides an example of the input that the remediation module expects to receive. Note: This value is not displayed in the web interface.	no

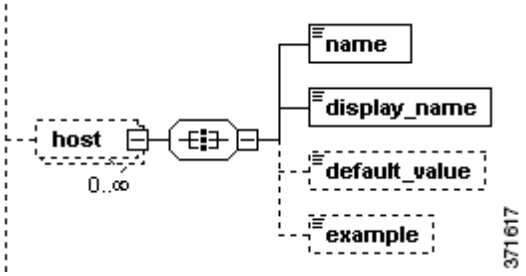
The following portion of a `config_template` element definition indicates that the web interface displays a field labeled “Netmask”, which accepts netmask values denoted by a dotted quad or CIDR mask and defaults to 255.255.255.255.

```
<netmask>
<name>netmask</name>
<display_name>Netmask</display_name>
<default_value>255.255.255.0</default_value>
</netmask>
```

The host Element

Each `host` element you use in a `config_template` represents a field in the web interface that accepts a single IP address or string.

The following diagram illustrates the child elements of the `host` element.



When configuring child elements for an occurrence of a `host` element, you may only use each available child element once. The following table describes the child elements and attributes available to the `host` element.

Table 8 host Attributes and Child Elements

Name	Type	Description	Required?
required	attribute	Indicates whether users must provide a value in the field. This attribute defaults to `true`. You are not required to use this attribute. Therefore, if you do not use it (or if you explicitly set its value to `true`), users must provide a value. If you set the value of the attribute to `false`, the web interface indicates that providing a value is optional.	no
`name`	element	Provides context to the remediation module for the value entered in the field. Names may not contain white space and may only contain alphanumeric characters and the underscore (_) and dash (-) character. Names should be unique within modules.	yes
`display_name`	element	Specifies the web interface label for this field.	yes
`default_value`	element	Specifies the default value for this field.If the web interface user does not specify a value, the remediation program uses this value by default.	no
`example`	element	Provides an example of the input that the remediation module expects to receive. Note: This value is not displayed in the web interface.	no

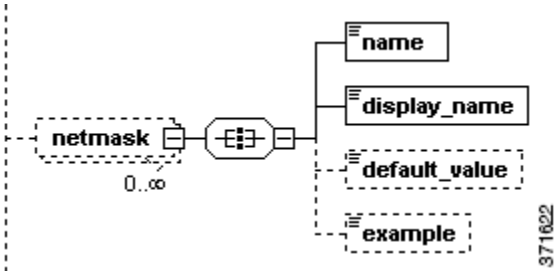
The following portion of a `config_template` element definition indicates that the web interface displays a field labeled “Host Name”, which accepts an IP address or string. The web interface also provides example text of “192.10.1.3.”

```
<host>
<name>hostname</name>
<display_name>Host Name</display_name>
<example>192.10.1.3</example>
</host>
```

The network Element

Each `network` element you use within a `config_template` represents a field in the web interface. A network field accepts an IP address (assumed to be a single IP address, that is, an IP address with /32 netmask) or a CIDR block.

The following diagram illustrates the child elements of the `network` element.



When configuring child elements for an occurrence of a `network` element, you may only use each available child element once. The following table describes the child elements and attributes available to the `network` element.

Table 9 network Attributes and Child Elements

Name	Type	Description	Required?
required	attribute	Indicates whether users must provide a value in the field. This attribute defaults to `true`. You are not required to use this attribute. Therefore, if you do not use it (or if you explicitly set its value to `true`), users must provide a value. If you set the value of the attribute to `false`, the web interface indicates that providing a value is optional.	no
`name`	element	Provides context to the remediation module for the value entered in the field. Names may not contain white space and may only contain alphanumeric characters and the underscore (_) and dash (-) character. Names should be unique within modules.	yes
`display_name`	element	Specifies the web interface label for this field.	yes
`default_value`	element	Specifies the default value for this field. If the web interface user does not specify a value, the remediation program uses this value by default.	no
`example`	element	Provides an example of the input that the remediation module expects to receive. Note: This value is not displayed in the web interface.	no

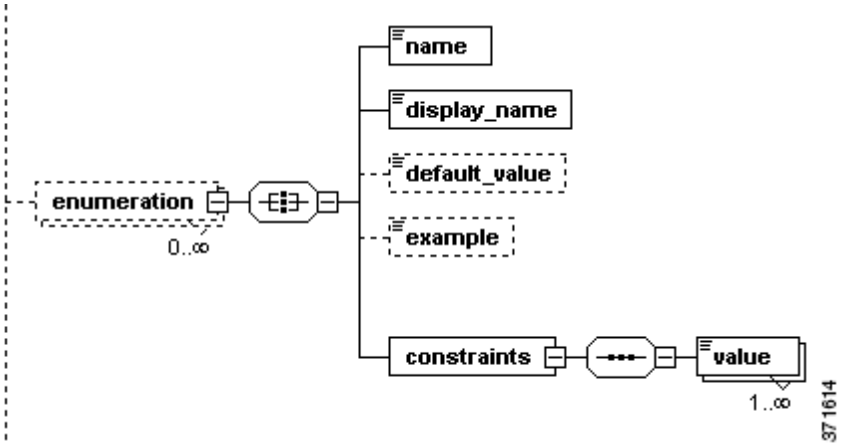
The following portion of a `config_template` element definition indicates that the web interface displays a field labeled “Monitored Network”, which accepts either a /32 IP address or an IP address and netmask value, and which has a default value of `192.168.1.0/24`.

```
<network>
<name>monitored_network</name>
<display_name>Monitored Network</display_name>
<default_value>192.168.1.0/24</default_value>
</network>
```

The enumeration Element

Each `enumeration` element you use in a `config_template` represents a drop-down list of strings displayed in the web interface. Users can select a single value from this list.

The following diagram illustrates the child and grandchild elements of the `enumeration` element.



The following table describes the child elements and attributes available to the `enumeration` element.

Table 10 enumeration Attributes, Child Elements, and Grandchild Elements

Name	Type	Description	Required?
required	attribute	Indicates whether users must provide a value in the field. This attribute defaults to `true`. You are not required to use this attribute. Therefore, if you do not use it (or if you explicitly set its value to `true`), users must provide a value. If you set the value of the attribute to `false`, the web interface indicates that providing a value is optional.	no
`name`	element	Provides context to the remediation module for the value entered in the field. Names may not contain white space and may only contain alphanumeric characters and the underscore (_) and dash (-) character. Names should be unique within modules.	yes
`display_name`	element	Specifies the web interface label for this field.	yes
`default_value`	element	Specifies the default value for this field. If the web interface user does not specify a value, the remediation program uses this value by default.	no

Table 10 enumeration Attributes, Child Elements, and Grandchild Elements (continued)

Name	Type	Description	Required?
`example`	element	Provides an example of the input that the remediation module expects to receive. Note: This value is not displayed in the web interface.	no
`constraints`	element	Specifies the values that the user can enter in this field. The `constraints` element has one required child element, `value`, that accepts a string that represents one choice for the users. Use multiple `value` elements to provide multiple choices to the user.	yes

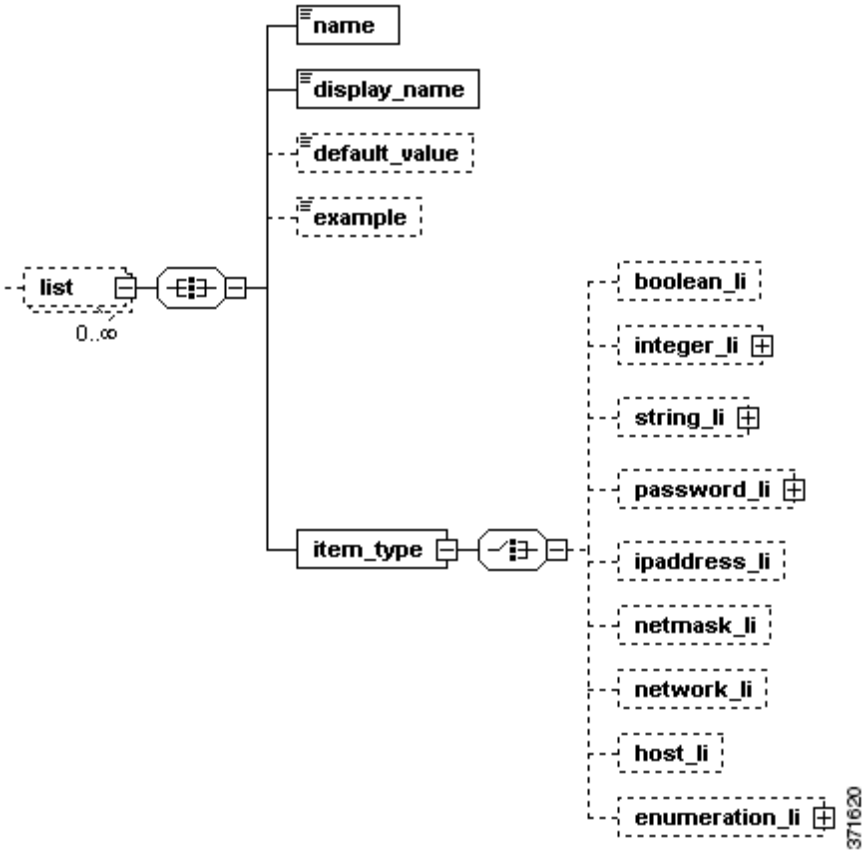
The following portion of a `config_template` element definition indicates that the web interface displays a field labeled “Day”, which allows users to select one of the values provided (`Monday`, `Tuesday`, `Wednesday`, `Thursday`, and `Friday`).

```
<enumeration>
<name>day</name>
<display_name>Day</display_name>
<constraints>
    <value>Monday</value>
    <value>Tuesday</value>
    <value>Wednesday</value>
    <value>Thursday</value>
    <value>Friday/value>
</constraints>
</enumeration>
```

The list Element

Each `list` element you use in a `config_template` represents a field in the web interface that allows users to enter a list of values, one per line, whose type is specified by the required `item_type` child element.

The following diagram illustrates the child and grandchild elements of the `list` element.



The following table describes the child elements available to the `list` element.

Table 11 list Attribute and Child Elements

Name	Type	Description	Required?
required	attribute	Indicates whether users must provide a value in the field. This attribute defaults to `true`. You are not required to use this attribute. Therefore, if you do not use it (or if you explicitly set its value to `true`), users must provide a value. If you set the value of the attribute to `false`, the web interface indicates that providing a value is optional.	no
`name`	element	Provides context to the remediation module for the value entered in the field. Names may not contain white space and may only contain alphanumeric characters and the underscore (_) and dash (-) character. Names should be unique within modules.	yes
`display_name`	element	Specifies the web interface label for this field.	yes
`default_value`	element	Specifies the default value for this field. If the web interface user does not specify a value, the remediation program uses this value by default.	no

Table 11 list Attribute and Child Elements (continued)

Name	Type	Description	Required?
example	element	Provides an example of the input that the remediation module expects to receive. Note: This value is not displayed in the web interface.	no
item_type	element	Specifies the type of value that can appear in this field. The value type is specified by a child element. Valid child elements are listed below.	no

The following list describes the child elements available to the item_type element, which are similar to the child elements of the config_template element; the only difference is that item_type child elements do not use the required attribute. Each instance of the item_type element can use only one child element:

- boolean_li indicates that the list accepts multiple Boolean values (see The boolean Element, page 3-6).
- integer_li indicates that the list accepts multiple integer values (see The integer Element, page 3-7).
- string_li indicates that the list accepts multiple string values (see The string Element, page 3-8).
- password_li indicates that the list accepts multiple password values (see The password Element, page 3-9).
- ipaddress_li indicates that the list accepts multiple ipaddress values (see The ipaddress Element, page 3-11).
- network_li indicates that the list accepts multiple network values (see The network Element, page 3-14).
- netmask_li indicates that the list accepts multiple netmask values (see The netmask Element, page 3-12).
- host_li indicates that the list accepts multiple host values (see The host Element, page 3-13).
- enumeration_li indicates that the list accepts multiple values as defined by the value child elements of the enumeration_li element's constraints child element (see The enumeration Element, page 3-15).

The following portion of a config_template element definition indicates that the web interface should allow the user to provide a list of integers between zero and 500 inclusive, one per line, in a field labeled "Integer List".

```
<list>
<name>list_integer</name>
<display_name>Integer List</display_name>
<example>Constrained value [0-500]</example>
<item_type>
   <integer_li>
      <constraints>
         <min>0</min>
         <max>500</max>
      </constraints>
   </integer_li>
</item_type>
```

```
</list>
```

Sample Configuration Template

This section provides a sample `config_template` element definition, which governs both the web interface appearance and the types of information the remediation module must receive from the user.

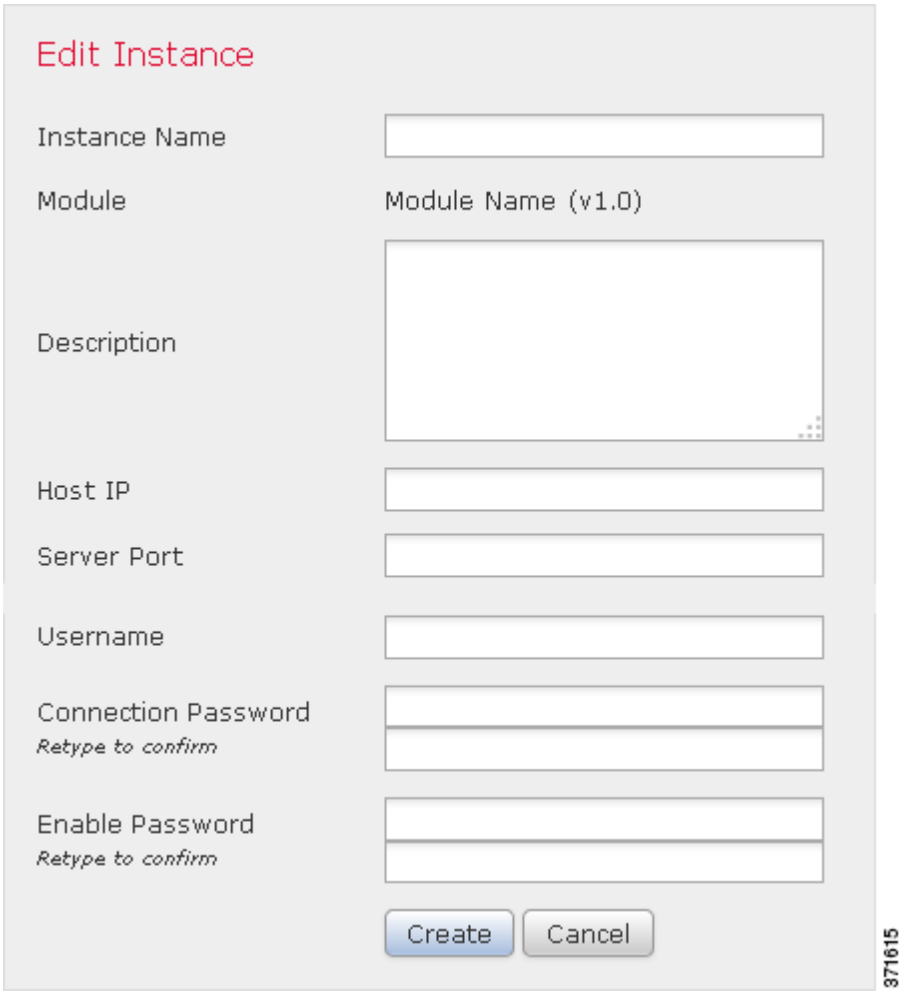
```
<config_template>
    <ipaddress>
        <name>host_ip</name>
        <display_name>Host IP</display_name>
    </ipaddress>
    <string>
        <name>user_name</name>
        <display_name>Username</display_name>
    </string>
    <password>
        <name>login_password</name>
        <display_name>Connection Password</display_name>
    </password>
    <password>
        <name>root_password</name>
        <display_name>Enable Password</display_name>
    </password>
</config_template>
```

The above template renders four fields on the web interface. The following table describes each field.

Table 3-12 Fields Created by the Sample Configuration Template

Field	Description
Host IP	Accepts an IP address that the remediation module identifies as `host_ip`.
Username	Accepts a string that the remediation module identifies as `user_name`.
Connection Password	Accepts an alphanumeric password string that the remediation module identifies as `login_password`.
Enable Password	Accepts an alphanumeric password string that the remediation module identifies as `root_password`.

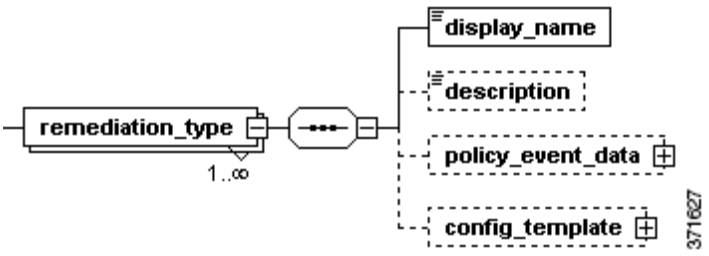
The following screen illustrates how these fields appear on the web interface. You must provide the data requested by these fields to configure the remediation module from the web interface.



Defining Remediation Types

Remediation types describe the actions, or *remediations*, taken by the device that is governed by the remediation module. Each `remediation_type` element you use in `module.template` represents one of those remediations. Remediations are triggered by correlation event data from the remediation subsystem. For more information see Event Data, page 2-2.

The following diagram illustrates the child elements of the `remediation_type` element.



The following table describes the attributes and child elements available to the `remediation_type` element.

Table 13 remediation_type Attributes and Child Elements

Name	Type	Description	Required?
`name`	attribute	Provides context to the remediation module for the remediation type. This attribute is required and accepts a string between 1 and 64 characters, inclusive. Names may not contain white space and may only contain alphanumeric characters and the underscore (_) and dash (-) character. `remediation_type` names must be unique within each module.	yes
`display_name`	element	Labels the remediation type on the web interface.	yes
`policy_event_data`	element	Specifies the correlation event data that the remediation module needs to receive from the remediation subsystem. The `policy_event_data` has one child element, `pe_item`, that represents a specific correlation event data item. Use multiple `pe_item` elements to provide multiple correlation event data items. For more information on appropriate correlation event data values, see Event Data, page 2-2.	no
`config_template`	element	Specifies the information the user must provide when configuring an instance of this remediation module. For more information, see Defining the Configuration Template, page 3-4.	no

The following portion of a `module.template` file illustrates several `remediation_type` element definitions.

```
<remediation_type name="block_src">
<display_name>Block Source</display_name>
<policy_event_data>
    <pe_item>src_ip_addr</pe_item>
    <pe_item>src_port</pe_item>
    <pe_item>src_protocol</pe_item>
</policy_event_data>
</remediation_type>
<remediation_type name="block_dest">
<display_name>Block Destination</display_name>
<policy_event_data>
    <pe_item>dest_ip_addr</pe_item>
    <pe_item>dest_port</pe_item>
    <pe_item>dest_protocol</pe_item>
</policy_event_data>
</remediation_type>
<remediation_type name="acl_insert">
<display_name>ACL Insertion</display_name>
<policy_event_data>
    <pe_item>src_ip_addr</pe_item>
    <pe_item>src_port</pe_item>
    <pe_item>src_protocol</pe_item>
    <pe_item>dest_ip_addr</pe_item>
    <pe_item>dest_port</pe_item>
    <pe_item>dest_protocol</pe_item>
</policy_event_data>
    <config_template>
        <integer>
            <name>acl_num</name>
```

```
            <display_name>ACL Number</display_name>
        </integer>
    </config_template>
</remediation_type>
```

The example above contains 3 remediation types: `block_src`, `block_dest`, and `acl_insert`. Each of these requires specific correlation event (`pe_item`) data. The `acl_insert` remediation type also requires configuration data, which is specified in its `config_template` child element; users must provide an ACL number when they configure instances of that type.

Defining Exit Statuses

The remediation subsystem expects to receive an exit status, or return code, in the form of an integer from your remediation module.

Cisco provides a set of predefined exit status messages your remediation module can return. You can return predefined exit statuses, which correspond to integer values between 1 and 128, inclusive. The following lists and describes these predefined exit status codes.

Table 3-14 Predefined Exit Statuses

Exit Status	Description
0	Successful completion of remediation.
1	Error in the input provided to the remediation module.
2	Error in the remediation module configuration.
3	Error logging into the remote device or server.
4	Unable to gain required privileges on remote device or server.
5	Timeout logging into remote device or server.
6	Timeout executing remote commands or servers.
7	The remote device or server was unreachable.
8	The remediation was attempted but failed.
10	A white-list match was found.
11	Failed to execute remediation program
20	Unknown/unexpected error.

Alternatively, your module may return integers between 129 and 254, inclusive, as custom exit statuses. If your remediation module returns custom exit statuses, you must define the set of exit statuses it can return. Each `exit_status` element you use in `module.template` represents a custom exit status that your remediation module can return. For more information, see Data Returned by Modules, page 2-12.

The `exit_status` element accepts a string that describes a return code. In addition, the element requires an attribute, `value`, that accepts a unique integer between 129 and 255. This attribute associates remediation module return codes with their descriptions, which the user can see in remediation status event views.

The following example illustrates valid custom `exit_status` elements.

```
<exit_status value="138">syslog error</exit_status>
<exit_status value="139">unknown error</exit_status>
```

Working with the Remediation SDK

Understanding the Remediation SDK

In addition to deploying Cisco-provided remediation modules, you can install and run your own custom remediations to automate responses to violations of associated correlation policies. Cisco provides a software developer kit (SDK) that you can download from the Support Site to help you get started.

Purpose of the SDK

Using the SDK and the information in this chapter of the Cisco Remediation API Guide, you can:

- Practice deploying a simple remediation module to gain familiarity with the process. Installation, configuration, and removal are easy.
- Inspect the source code of a remediation program to see one way to use the API to interact with the remediation subsystem and perform multiple remediation functions.

Caution: The syslog module in the SDK is not intended for production use.

Please note that you can use the Cisco-provided modules already loaded on the Firepower Management Centeras a reference resource while you develop. All of these modules are accessible at `/var/sf/remediation_modules` on the Firepower Management Center. Each installed module has a `.tgz` package in this directory. For information on the modules, see Cisco-Provided Remediation Modules, page 1-3.

Description of the SDK

The remediation SDK has a syslog alert remediation module in two versions, Perl and C. To use it, you need a syslog server running and receiving remote traffic.

The module provides two remediation types:

- `Simple_Notification` - generates syslog alerts with the source IP address, source port (if available), and IP protocol (if available) for the triggering event.
- `Complete_Notification` - generates a syslog alert with the same fields as the simple notification, and also includes the destination IP address, destination port, and a severity indicator for the triggering event.

As with all remediation modules, you enter a small amount of configuration in the web interface to add instances of the module. Each instance targets a particular device on your network (in this case a syslog server) and runs the remediation for the instance. To run the `Complete_Notification` remediation type, you select a syslog facility level not required for the `Simple_Notification` remediation type.

See the following table for a list of the Perl version files.

Table 4-1 Sample Perl Module

Included Files	Description
syslog.pl	The program that executes the syslog alert when the correlation policy associated it with is violated.
module.template	Module configuration file. Defines required event data, required information to collect in the web interface when users create instances, and other essential setup parameters.
Makefile	Sample makefile to package the files in a remediation module for installation on the Firepower Management Center.

See the following table for a list of the C version files.

Table 4-2 Sample C Module

Included Files	Description
syslogc.c	The program that executes the syslog alert when the correlation policy associated it with is violated.
module.template	Module configuration file. Defines required event data, required information to collect in the web interface when users create instances, and other essential setup parameters.

Downloading the SDK

To download the remediation SDK:

1. Access the support website at https://www.cisco.com/c/en/us/support/index.html
2. Select **Security**, then select **Firepower Management Center**. and select **Download Software**.
3. Select any of the Management Center options.
4. Download the **CISCO FIREPOWER MANAGEMENT CENTER REMEDIATION MODULE FOR ACI**
5. Unpack the .zip file in a convenient folder on your client machine.

Overview of the Development and Installation Process

The steps below form a checklist of tasks that need to be performed to create, install, and configure a custom remediation module. Some of the steps involve procedural and descriptive details that are explained in cross-referenced sections of the *Remediation API Guide* or the *Firepower System User Guide*.

To develop, install, and configure a custom remediation module, you must:

1. Identify the condition you want to mitigate and the actions that appropriately resolve the detected condition in your environment.
2. Familiarize yourself with data elements that can be obtained from the remediation subsystem. See Data Available from the Remediation Subsystem, page 2-1 for definitions of all available fields that the Firepower Management Center can provide for your remediation.

You should also understand the return code functionality built into the remediation subsystem. See Defining Exit Statuses, page 3-22 for information.

3. Generate a high-level design that identifies all the remediation actions (remediation types) that your program needs to address.
4. Write your remediation program so that it addresses all the functions necessary for the desired remediations. Remediation module programs may be written in bash, tsch, Perl or C. Develop your program using the technical guidance provided in Notes for Remediation Program Developers, page 4-3.
5. Create the module template file for your remediation module. For an understanding of the data elements and syntax of the module template, see the chapter Communicating with the Remediation Subsystem, page 3-1.

You can save time by editing an existing `module.template` file to start with.

6. Package your remediation module as described in Packaging Your Module, page 2-12.
7. Install the module on the Firepower Management Center using the Policy and Response component as described in Installing Your Module, page 2-13. You will load the package on the Firepower Management Center and proceed as if you were configuring one of the Cisco-provided modules.
8. Ensure that the individual remediation types in your remediation module are assigned as responses to the correct correlation rules in your defined correlation policies. See the *Firepower System User Guide* for procedure details.

Notes for Remediation Program Developers

When you have defined the required scope and functionality of your remediation program and understood the data elements available for your remediation actions, you can write the remediation program.

Remediation module programs may be written in bash, tsch, Perl or C.

The following table indicates where to find information on topics of interest.

Table 4-3 Programmer Notes

To learn more about...	Look in...
the file structure and workflow environment of the remediation subsystem	Understanding the Remediation Subsystem File Structure, page 4-4
implementing multiple remediation types in a remediation program	Implementing Remediation Types in a Remediation Program, page 4-4
the remediation subsystem file structure	Understanding the Remediation Subsystem File Structure, page 4-4
the interactions of the remediation program and the Firepower Management Center remediation subsystem	Understanding the Remediation Program Workflow, page 4-5
the order in which parameters are passed from the Firepower Management Center to the remediation module	The Order of Command Line Parameters, page 4-5
how the remediation daemon handles undefined data elements	Handling Undefined Data Elements, page 4-6
return codes from the remediation program	Handling Return Codes, page 4-6

Table 4-3 Programmer Notes (continued)

To learn more about...	Look in...
runtime modes for the remediation program	Important Global Configuration Elements, page 4-6
alternative encoding of user input	Important Global Configuration Elements, page 4-6

Implementing Remediation Types in a Remediation Program

The remediation daemon on the Firepower Management Center specifies the remediation name as the first argument on the command line when it launches the remediation program. The code snippet below from the SDK Perl program, `syslog.pl`, shows one way your program can branch to the appropriate remediation function. The program runs either `SimpleNotification()` or `CompleteNotification()`, based on the content of `$remediation_config`, which is set by the first field from the remediation daemon. The sample also shows the use of return codes which are discussed in Handling Return Codes, page 4-6.

```
# Call the appropriate function for the remediation type
my $rval = 0;
if($remediation_config->{type} eq "Simple_Notification")
{
$rval = SimpleNotification($instance_config, $remediation_config,
\@pe_event_data);
}
elsif($remediation_config->{type} eq "Complete_Notification")
{
$rval= CompleteNotification($instance_config,$remediation_config,
\@pe_event_data);
}
else
{
warn "Invalid remediation type. Check your instance.conf\n";
exit(CONFIG_ERR);
}
exit($rval);
```

You declare the names of all remediation types in the `module.template` file, and associate remediation types with each instance as you add the instance via the web interface. The remediation type that is executed by the instance is recorded in the `instance.config` file which is stored in the `instance.config` subdirectory described in Understanding the Remediation Subsystem File Structure, page 4-4.

Understanding the Remediation Subsystem File Structure

The root directory of each remediation module is derived from the remediation module name and version number, both of which are declared in the `module.template` file. See The config Element, page 2-9 for details on the elements of `module.template`.

If you install a module packaged in syslog.tgz with the name `syslog` and version `1.0` in `module.template`, the system puts the module in the following directory: `/var/sf/remediation/syslog_1.0`. That directory contains the `module.template` file and the remediation program binary for the module.

When you add an instance of the remediation and name the instance `log_tokyo`, the system creates the following directory:

```
/var/sf/remediation/syslog_1.0/log_tokyo
```

and places a file named `instance.conf` in it. The `instance.conf` file, which is in XML format, contains the configuration information for the `log_tokyo` instance.

The following Linux command sequence illustrates the directory structure described above.

```
# cd /var/sf/remediations
# ls
NMap_perl_2.0  SetAttrib_1.0         cisco_pix_1.0
cisco_ios_router_1.0  syslog_perl_0.1
# cd syslog_perl_0.1
# ls
log_chicago log_tokyo module.template syslog.pl
# cd log_tokyo
# ls
# instance.conf
```

Note that the `instance.conf` file contains the name of the remediation type that the `log_tokyo` instance runs. In the above example, the user who added the `log_tokyo` instance could have configured it to run either remediation type defined for the syslog remediation module: `Simple_Notification` or `Complete_Notification`.

For details on the elements in the instance.conf XML file, see Instance Configuration Data, page 2-8.

Understanding the Remediation Program Workflow

When the Firepower Management Center executes a remediation instance, the remediation daemon launches the remediation program from the instance subdirectory and supplies data from the `instance.conf` file to the remediation program as command line arguments.

An example will illustrate the process. If a policy violation launches a syslog instance named `log_tokyo`, which calls the remediation named Simple_Notification with a source IP address of 1.1.1.1 and a destination IP address of 2.2.2.2, the Firepower Management Center sets the working directory to `/var/sf/remediations/Syslog_1.0/log_tokyo` (that is, the `instance.conf` subdirectory) and executes the remediation binary, `syslog.pl`. The daemon's command line syntax will be as follows:

```
../syslog.pl Simple_Notification 1.1.1.1 2.2.2.2
```

Note in particular that the `syslog.pl` executable is in the parent directory of the `instance.conf` subdirectory.

When the command runs in this way, the `syslog.pl` binary can load the information in `instance.conf` file because it is in the current directory. If the binary needs to load any modules or other files in the parent directory (`/var/sf/remediations/Syslog_1.0` in this case), the code must explicitly load them from the parent directory; that is, it must provide a path starting with `"../"`. Otherwise the binary will not be able to find the files that it needs.

In Perl, you can also deal with this issue using the `lib()` function as follows:

```
use lib("../");
```

Your program must be able to open, read, parse, and close the `instance.conf` file.

The Order of Command Line Parameters

When the remediation daemon passes event data to your remediation module, it passes the name of the remediation followed by the correlation event data in the order in which the fields are specified in `module.template`. In `module.template`, each field to be passed to your module is declared using the `<pe_item>` tag.

If a `pe_item` is set to optional in `module.template` and is undefined (meaning there is no value for the specific `pe_item`), the remediation daemon passes “undefined” or null to your module. If `pe_item` is set to required in `module.template` but is undefined, the remediation daemon logs a message to the remediation log stating that no value is available, and does not execute your remediation module binary. You can view the remediation log in the web interface where it is called the Table View of Remediations. See the *Firepower System User Guide* for details on how to access and use this view.

Handling Undefined Data Elements

The remediation daemon handles undefined data items differently, depending on whether an item is marked as `optional` or `required` in `module.template`. Undefined means that the Firepower Management Center database has no value for the item. The daemon’s processing is as follows:

- If the undefined `pe_item` is set to `optional` in `module.template`, the daemon passes “undefined” or null to your module.
- If the undefined `pe_item` is set to `required` in `module.template`, the daemon does not execute the remediation and logs a message to the remediation log stating that no value is available.

Handling Return Codes

The Firepower Management Center waits for a return code for each instance and records the code in the remediation log. For information on predefined and custom return codes, see Defining Exit Statuses, page 3-22.

The Table View of Remediations in the web interface of the Firepower Management Center displays the results of each launched remediation. See the *Firepower System User Guide* for information on accessing and using the Table View of Remediations.

Important Global Configuration Elements

You can enable the remediation API features described in the table below by setting their corresponding elements in the `module.template` file. For configuration details, see Defining the Global Configuration, page 3-2.

Table 4-4 Features Enabled in Global Configurations of module.template

To enable this feature...	Set this module.template parameter ...
run remediation program as root	run_as_root Warning: Cisco recommends that you use this element only if absolutely necessary.
HTML-encoding of user input	encode_values Note: If you use this element, your remediation module must handle HTML decoding as part of its input handling.