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 Defense Center as a reference resource while you develop. All of these modules are accessible at `/var/sf/remediation_modules` on the Defense Center. Each installed module has a `.tgz` package in this directory. For information on the modules, see Cisco-Provided Remediation Modules, page 1-2.

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 Defense 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://support.sourcefire.com/downloads.
2. Select a software version, then under Product Category, select **Software**. The download link for the remediation SDK is in the **api** portion of the page.
3. 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 *FireSIGHT 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 Defense 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 2-13.
7. Install the module on the Defense Center using the Policy and Response component as described in Installing Your Module, page 2-13. You will load the package on the Defense 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 *FireSIGHT 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 Defense Center remediation subsystem	Understanding the Remediation Program Workflow, page 4-5
the order in which parameters are passed from the Defense 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 4-5
return codes from the remediation program	Handling Return Codes, page 4-6
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 Defense 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 2-9.

Understanding the Remediation Program Workflow

When the Defense 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 Defense 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 *FireSIGHT 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 Defense 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 Defense 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 Defense Center displays the results of each launched remediation. See the *FireSIGHT 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 3-3.

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 Note: Warning: Cisco recommends that you use this element only if absolutely necessary.
HTML-encoding of user input	encode_values Note: Note: If you use this element, your remediation module must handle HTML decoding as part of its input handling.