|  |  |  |  |
| --- | --- | --- | --- |
| **Spec Name:** |  | | |
| **Feature** | Feature #0855: Automatic Remediation | | |
| **Bug Fix/CR #(s):** | N/A | | |
| **Associated Release** | 6.0.Beta | | |
| **Developer(s):** | Chris Petersen, Phillip Dressen, Brian Staton (primary), Phillip Villella | | |
| **Date Created:** | 7/13/2010 | **Last Date Updated:** | 8/11/2011 |
| **Associated Software Programs:Components** | ARM: Automatic Remediation Engine, Console: Automatic Remediation Plugin Manager, Console: Alarm Viewer | | |

Note – this was only a conceptual design spec but Phil D updated it as the design note instead of creating a new one. It is missing the supplementary sections for this reason.

Updated History:

Phillip Dressen Dec 20, 2010 – Initial Draft

Phillip Villella August 12, 2011 – Changes to bring spec into agreement with EngNote\_AutoRemediationFixes\_B4825.docx

# Background

Generally Automatic Remediation (AR) is the ability for a sensor-based technology to take automatic corrective action resulting from observed events. Actions can be initiated in a variety of ways. For instance, TCP Resets can be issued to terminate a connection on the network or commands may be issued against a Windows Domain Controller to disable a user account. Automatic Remediation is typically implemented in Intrusion Detection, Intrusion Prevention, and Security Event Management products. Automatic Remediation has had mixed results in terms of field-based success. Often the objective of automatic remediation doesn’t align well with IT change control process and users end up being uncomfortable or unable to let technology automatically take corrective action within the IT environment. Therefore full automatic remediation is rarely aggressively implemented even though it can be a principle buying requirement. It is a feature that can be seen as critical at time of purchase but often never implemented due to real-world IT operations constraints.

LogRhythm’s general direction with Automatic Remediation will be to provide effective remediation capabilities that satisfy initial purchase requirements and provide flexible remediation options that can interface with a variety of devices via in-field integration. We will also implement a model of “Managed Automatic Remediation” in which LogRhythm can automatically perform the action, but requires user input before doing so. To make LogRhythm’s AR as extensible and flexible as possible, a “plug-n-play” architecture will be pursued that will allow users, Partners and LogRhythm Professional Services engineers to develop AR capabilities via “plug-in” programs and scripts that can be developed outside the LogRhythm core code base. This is counter to developing an architecture where LogRhythm develops specific interfaces and remediation methods directly within the AR code base. This “plug-in” approach should result in an AR capability to can be configured to initiate automatic or user driven “managed” remediation by executing a specified “plug-in” program or script that can accept input parameters via one or more standard input mechanisms. This will allow customers to leverage existing programs and scripts and allow LogRhythm to develop new remediation “plug-ins” without having to modify the AR code base itself.

The AR capabilities will be implemented within the ARM service. AR is similar to the existing Notification Engine within the ARM. Whereas the Notification Engine sends messages resulting from Alarms, the AR Engine will send instructions to “plug-ins” to execute a remediation action. Upon implementation, a path may emerge where AR is an extension of the existing notification engine capabilities.

Automatic Remediation actions will be contained within an Automatic Remediation Rule (ARR) that encapsulates configuration settings. ARRs will be directly associated with an Alarm Rule, similar to current Notification settings. For Harmony Rules, ARRs will be associated to the Alarm Rule that is automatically associated with a Harmony Rule. This will allow Automatic Remediation to function with existing Alerting capabilities and those being introduced via Harmony since the ARM will continue to be responsible for Alarm generation, Notifications and going forward, Automatic Remediation.

# Key Requirements

The following are key requirements for the implementation of Automatic Remediation in LogRhythm 6.0. Listed are mandatory requirements which must be met and optional requirements that would be nice to meet. Optional requirements should be designed towards since these capabilities will most likely be introduced in subsequent releases.

## Mandatory

* Remediation actions can be initiated from the Event Manager ARM service.
* Actions can be performed leveraging existing Windows-based programs and scripts. Program/script parameters and data-input parameters can be flexibly defined and passed to the program/script to ensure specific execution of the desired action.
* Actions can be configured to be performed automatically or require user confirmation before the action will be initiated.
* Actions can be configured such that more than 1 individual must approve before being executed (i.e., Network Administrator + Director IT)
* Actions can be configured to use data from the alarm and the first event associated with that alarm.
* Actions must be started within max latency of 60 seconds of the Alarm being generated. Ideal target is to reduce latency to around 1 minute.
* Actions will be encapsulated in a Rule that is associated with the existing Alarm Rule and is seamlessly presented to the user as part of the existing Alarm Rule Properties.
* Actions will only be available for Global Admin Alarm Rules. Private Alarm rules will not support remediation.
* A mechanism needs to exist for testing a configured Action.
* In the UI, the Alarm Record will be updated to display Remediation Approvals, Actions and associated status.
* In the UI, the Alarm Viewer will be updated to display a counter in the list that indicates the number of Remediation Actions pending approval, denied, successfully executed, or failed.
* In the UI, the Alarm Viewer will be updated to visually display if an Alarm has pending remediation actions that require confirmation.
* In the UI, the Alarm Viewer will expose a form that allows the user to approve one-or-more pending remediation actions in batch.

## Out of Scope (Optional or Future Work)

1. Remediation actions can be initiated from a Windows System Monitor
2. “System” Remediation Actions will be supported so that the action itself and all related objects can be imported via a Knowledge Base enabling the development and delivery of Alarm Rules that contain pre-configured Automatic Remediation support.
3. Alarm Reporting will be updated to include Automatic Remediation Approvals and Action activity.
4. Remediation actions can be initiated from a \*Nix System Monitor leveraging \*Nix-based programs and scripts.
5. When testing a remediation action, the user can configure a “roll-back” operation to set back to original state following test.
6. Actions performed will be based on the unique events associated with an Alarm Rule. Because an Alarm Rule may have multiple associated events, multiple actions may be required per Alarm. Furthermore, additional Remediation actions may be required as additional Events are associated with the Alarm over time.

# Conceptual Design

## Automatic Remediation Plug-In (ARPI)

ARPIs are a zipped package (specifically tar archive plus gzip) containing executables, scripts and a Configuration file. The configuration file describes Remediation Actions that can be executed using the executables and scripts packaged in the plug-in.

The internal structure of the plug-in consists of three folders:



Plugin - The container that holds everything related to the plug-in.

bin – The folder containing all folders and files that are distributed with the plug-in folder for execution. Because a plug-in may leverage existing Windows commands, this folder may be empty. When an action within the plug-in is executed, the plug-in will be unzipped to a temporary folder and the command is executed from within this folder. So commands that reference an internal folder hierarchy should use this point as the relative location for execution.

config – The folder containing the XML Configuration File “actions.xml”. This configuration file must be present to define the Remediation Actions available with this plug-in.

### Plug-in Creation

To create a plug-in the user must already have a configuration file defined, and the appropriate executables and scripts ready. Using LogRhythm, the user can package the plug-in and save it to a file for importing. To create the plug-in the user specifies the configuration file, and a listing of Files and Directories to include (everything below a directory will be included in the plug-in). When a user is ready to create the plug-in they will be prompted for a file name and location to save it to. By default the file extension for remediation plug-ins is “.lpi”.

Optionally, the user can choose to omit the values for GUID and Version from the plug-in configuration file. If these values are not supplied then a GUID will be generated and the version will be set to 1.

The plug-ins will execute commands with zero or more command line arguments. The user will have no further opportunity to interact with any application once it has been started, so during the process of plug-in design it is important to verify that a plug-in application can execute successfully by receiving all of its information from command line arguments.

Developer Note: The logic for creating a plug-in can be located within the lrautormdn project within the class ‘PluginPackageCreator’.

### Plug-in Importing and Action Registration

Plug-ins may be imported either when they are created by the user as described in the ‘Plug-in Creation’ section or a plug-in may be imported as a file. The importer populates two tables as a result of the contents of the plug-in. The first table is the EMDB.AutoRmdnPlugin table, which holds a complete copy of the plug-in which is required when an action from this plug-in is executed. The second table is the EMDB.AutoRmdnRegisteredAction table, which is populated from the contents of the configuration file within the imported plug-in. Inside this configuration file is specification for every action that this plug-in supports. Each action is imported in to the registered action table so they can be referenced individually when a user is configuring an action on an Alarm Rule.

Currently all plug-ins and registered actions (database tables AutoRmdnPlugin and AutoRmdnRegistered action respectively) are imported into the ‘User’ ID space, to allow for the future option of defining LogRhythm System Plug-ins.

Plug-in Versioning (discussed in a separate section), has a couple of impacts when importing the newer version. First, the new plug-in is imported into the database making it available for configuration, next all the actions on the older version of the plug-in (if any) are reassigned to use the new plug-in. Finally, the older version is retired to indicate that it is no longer in use and to prevent users from configuring new actions on the older plug-in.

Developer Note: Plug-in Importing and Registration logic are in the lrautormdn project within the class ‘PluginPackageImporter’

### Plug-in Configuration File

As described in the Automatic Remediation Plug-in section, there is a required configuration file within every plug-in. This configuration file exists for the purpose of defining actions that can be taken in response to Alarms. If the action is not defined in the configuration file of a plug-in, then it cannot be registered and subsequently configured on an Alarm Rule or executed.

The configuration file defines a set of actions within a plug-in. Those actions contain the definitions for a command line action that should be executed as a remediation action. So those actions may contain parameters which define the command-line arguments associated with a command. Those parameters may or may not define input values, which allow the action to be further configured either by the user (when it is configured on an Alarm) or by the Alarm/Action when it is executed.

Developer Note: The XML configuration file is constructed as a ‘serialized’ representation of a set of objects within the remediation feature. The class structure used for describing plug-ins and actions is located in the lrautormdn project under the Configuration folder (namespace: SC.lrautormdn.Configuration).

#### Validation

<remv1:Remediation-Plugin **xmlns:remv1="RemediationVersion1.xsd"** …

Every configuration file is supplied in XML format, which means that to pass validation it must first qualify as a ‘well-formed’ XML document. Primarily this means that there is exactly one root tag, and every tag has a matching closing tag (as defined by XML). Additionally, LogRhythm Remediation Configuration files are examined for content by forcing the documents to conform to a defined XSD, which defines available tags, attributes, number and construction. In order to comply with the XSD document used for validation, configuration files need to define that file with an XML Namespace as shown in the ‘Remediation-Plugin’ tag above.

#### Remediation Plug-in (Configuration)

<remv1:Remediation-Plugin xmlns:remv1="RemediationVersion1.xsd" **Name=**"Plugin Name" **Guid=**"00000000-0000-0000-0000-000000000000" **Version=**"1">

<…>

</remv1:Remediation-Plugin>

A remediation plug-in configuration object defines the contents of everything associated with the plug-in. This includes the Name this plug-in is associated with, the GUID used for identifying this plug-in uniquely, the Version of the plug-in (“1” for anything created outside of LogRhythm) and every action this plug-in package is intended to support.

As noted in the Plug-in Creation section, the user can optionally omit the attributes for GUID and Version from the configuration. These values will be supplied and available within the configuration file created during plug-in creation.

<remv1:Remediation-Plugin xmlns:remv1="RemediationVersion1.xsd" Name**=**"Plugin Name" Guid**=**"00000000-0000-0000-0000-000000000000" Version**=**"1" **IsLogRhythmPlugin**=”true”>

<…>

</remv1:Remediation-Plugin>

Optionally, the attribute “IsLogRhythmPlugin” can be used to identify plug-ins created internally, also enabling the ability to Version the plugins. This attribute should not be documented for external consumption.

#### Action

<remv1:Remediation-Plugin … >

**<remv1:Action** **Name=**"Cmd Command" **Command=**"powershell.exe"**>**

<…>

**</remv1:Action>**

</remv1:Remediation-Plugin>

An action defines the particular unit of a single remediation action to be executed. This is the definition of what will result in a command-line execution of a script or executable and its associated command-line arguments. The action itself requires two attributes: Name and Command. Command defines precisely the executable that runs as a process to complete the action. In the case of scripts, this is the executable that understands how to run the script that must be provided as a parameter, such as the example above where the command is “powershell.exe”. The Name is simply the succinct name by which this individual command can be referenced within LogRhythm Console. There is no requirement for uniqueness among action names globally (because it is impossible to enforce), however it is advisable to provide action names that are uniquely identifying within the plug-in.

For identification purposes, action names must be unique within a plug-in. This is enforced both during plug-in creation and plug-in importing.

#### Parameters

Each parameter defines a single argument that will be supplied to the command-line arguments of a process that is executed as the result of a remediation action. Parameters may be defined as one of three types: Constant, String and Encrypted, each is outlined in greater detail below. The main distinction between the parameter types is dependent upon the context of how the action should be configured and used. Constant parameters are command-line arguments that should not change for this action, regardless of the desires of the user or the context of the Alarm that was raised. String parameters are command-line arguments that may be different each time an Action is executed depending upon how the action is configured on an alarm (see Input Values). Finally, Encrypted Parameters modify the behavior of a String Parameter so that the value is stored with some additional protection.

The attribute Name supplied for Constant and String Parameters must be unique within an action, but it is acceptable to reuse the name inside other actions within the same plugin.

##### Constant Parameters

<remv1:Action … Command="powershell.exe">

**<remv1:ConstantParameter Name=”**DisplayName**” Switch=**"-File Sample.ps1"Order=”Unsorted”**/>**

</remv1:Action>

Constant Parameters remain the same for a Remediation Action regardless of how it is configured by the user or the alarm that triggers the action. These are useful for fixing a command line parameter within the context of a particular action.

A good example of this is with an action that intends to execute a PowerShell script called “Sample.ps1”. The full command line for this would be [powershell.exe –File Sample.ps1]. As discussed previously, the action would have Command set to “powershell.exe” as this is the process to be executed. The parameter here would be set to the complete value “-File Sample.ps1”. The only tasks required to complete this configuration, are to give the action a name and place it within the tag for a remediation plug-in configuration.

##### String Parameters

<remv1:Action …>

**<remv1:StringParameter Name=”** DisplayName**” Switch="**\address:" Order=”Unsorted”**/>**

</remv1:Action>

Ultimately, the downside to Constant Parameters is the inability to alter the action taken based upon values the user or alarm can provide. This is what the String Parameter provides.

At first glance the configuration of a Constant Parameter and String Parameter look nearly identical. For a hypothetical example, pretend the full command line is [xyz.exe \address:”127.0.0.1”] and “127.0.0.1” can actually be a value that is set by a user when the action is configured on an Alarm Rule, or possibly SIP or DIP from the Alarm or one of its events. In contrast to the Constant Parameter, that used the entire argument for the Switch, the String parameter only uses “\address:” the prefixed portion of the argument used to identify the meaning of the value that is provided afterwards.

It is critical that the ‘Switch’ attribute contains the complete switch including the switch identifier (most commonly ‘\’ or ‘-‘) and the delimiter (most commonly ‘:’ ‘=’ or a space). Command-line programs vary widely on how well they support variation in the switch identifier and delimiter, and the only way to ensure that the value provided matches the executable’s expectations is to require the information in the parameter definition.

In order to finally execute a remediation action, all String parameter must completely resolve their argument. Internal to Remediation this is accomplished by setting the ‘Input’ of a String Parameter when it is configured on an Alarm Rule. Afterwards the final value of the command-line argument is determined by the options used on Input (see Section “Input Values”). To make the process of assigning values to String Parameters when setting up an Alarm Rule the String Parameter does provide the option of assigning a ‘Default Input Value’, again the section on “Input Values” provides greater detail.

##### Encrypted Parameters

<remv1:Action …>

**<remv1:EncryptedParameter>**

<remv1:StringParameter Switch="\switch:" Order=”Unsorted”/>

**</remv1:EncryptedParameter>**

</remv1:Action>

An encrypted parameter, only identifies another parameter as one whose value contains sensitive information (such as a password, community string, or other information). When a parameter is identified as encrypted, any value assigned to that parameter is encrypted when it is stored. Since this is simply a modification of the behavior provided by String Parameter, an Encrypted Parameter has no options of its own and simply wraps an existing String Parameter.

It is suggested that Encrypted Parameters should be used sparingly (not everything is sensitive). Parameters that are intended to use information from an Alarm/Event do not require encryption because the information is already provided in plaintext. Additionally, it is recommended that Encrypted Parameters are not defined with a Default Input Value, because the configuration file is already stored in plaintext within the plug-in.

##### Input Values

Input Values can be one of two types: Static Input or Metadata Input. Static Input are values that remain the same for each execution of an action against a particular Alarm Rule. Metadata Input values vary based upon the contents of the Alarm triggering the Remediation Action. In order to configure an Action on an Alarm Rule every String and Encryption Parameter must have an Input Value set to its Input Property. This occurs when the user defines a Remediation Action on an Alarm Rule.

Developer Note: An ‘Input Value’ is defined by object implementing the IRemedationInputValue interface.

##### Static Input Values

A Default Input defines a way to specify a Static or Metadata Input on a String Parameter from within the Configuration File. This does not actually set the Input Property of that String Parameter for Execution, but a default that the UI can use to populate the Input Value by default.

<remv1:StringParameter …>

<remv1:DefaultInput>

<remv1:StaticInput Value="DefaultInputValue" />

</remv1:DefaultInput>

</remv1:StringParameter>

Notice that a String Parameter with a Static Default Input Value is very similar to using a Constant Parameter. The primary exception is that the Default Input only acts as a suggestion to how the argument is completed, the user still controls actual value used when the action is configured on an Alarm Rule. If the user accepts the default, then this will construct a command-line argument as [\switch:”DefaultInputValue”].

##### Metadata Input Values

Metadata Values are populated from the Alarm that generates an action. Values come from the Alarm itself and the first event associated with the alarm. Just like the Static Input Value, Default Input can be defined in the configuration for Metadata Input Values.

<remv1:StringParameter …>

<remv1:DefaultInput>

<remv1:**CommonEvent** />

</remv1:DefaultInput>

</remv1:StringParameter>

Each field of the alarm or event has its own tag that can be used to specify the intended value. In the example above the default value for the String Parameter will be populated with the CommonEvent of the first action associated with the Alarm. Outlined in the table below is the list of supported fields that can be used for Metadata Input Values.

| Field | Alarm or First Event | Example Tag | Format Support |
| --- | --- | --- | --- |
| AlarmID | Alarm | <remv1:**AlarmId** /> |  |
| AlarmDate | Alarm | <remv1:AlarmDate>  <remv1:TimeFormat TimeZone="Eastern Standard Time" FormattingString="MMMM dd, yyyy" />  </remv1:AlarmDate> | Date and Time |
| AlarmRuleName | Alarm | <remv1:AlarmRuleName /> |  |
| MARCMsgID | First Event | <remv1:MARCMsgId /> |  |
| Direction | First Event | <remv1:Direction /> |  |
| NormalMsgDate | First Event | <remv1:NormalMsgDate>  <remv1:TimeFormat TimeZone="Eastern Standard Time" FormattingString="MMMM dd, yyyy" />  </remv1:NormalMsgDate> | Date and Time |
| MsgClass | First Event | <remv1:MessageClass /> |  |
| CommonEvent | First Event | <remv1:CommonEvent /> |  |
| MPERule | First Event | <remv1:MPERule /> |  |
| KnownSHost | First Event | <remv1:KnownSHost /> |  |
| KnownDHost | First Event | <remv1:KnownDHost /> |  |
| KnownService | First Event | <remv1:KnownService /> |  |
| SIP | First Event | <remv1:SIP /> |  |
| DIP | First Event | <remv1:DIP /> |  |
| SHostName | First Event | <remv1:SHostName /> |  |
| DHostName | First Event | <remv1:DHostName /> |  |
| SPort | First Event | <remv1:SPort /> |  |
| DPort | First Event | <remv1:DPort /> |  |
| Protocol | First Event | <remv1:Protocol /> |  |
| Login | First Event | <remv1:Login /> |  |
| Account | First Event | <remv1:Account /> |  |
| Sender | First Event | <remv1:Sender /> |  |
| Recipient | First Event | <remv1:Recipient /> |  |
| Subject | First Event | <remv1:Subject /> |  |
| Object | First Event | <remv1:Object /> |  |
| VendorMessageID | First Event | <remv1:VenderMessageId /> |  |
| BytesIn | First Event | <remv1:BytesIn /> |  |
| BytesOut | First Event | <remv1:BytesOut /> |  |
| ItemsIn | First Event | <remv1:ItemsIn /> |  |
| ItemsOut | First Event | <remv1:ItemsOut /> |  |
| Duration | First Event | <remv1:Duration /> |  |
| Process | First Event | <remv1:Process /> |  |
| Amount | First Event | <remv1:Amount /> |  |
| Quantity | First Event | <remv1:Quantity /> |  |
| Rate | First Event | <remv1:Rate /> |  |
| Size | First Event | <remv1:Size /> |  |
| Domain | First Event | <remv1:Domain /> |  |
| Group | First Event | <remv1:Group /> |  |
| URL | First Event | <remv1:URL /> |  |
| Session | First Event | <remv1:Session /> |  |
| SNetwork | First Event | <remv1:SNetwork /> |  |
| DNetwork | First Event | <remv1:DNetwork /> |  |
| SLocationCountry | First Event | <remv1:SLocationCountry /> |  |
| SLocationRegion | First Event | <remv1:SLocationRegion /> |  |
| SLocationCity | First Event | <remv1:SLocationCity /> |  |
| DLocationCountry | First Event | <remv1:DLocationCountry /> |  |
| DLocationRegion | First Event | <remv1:DLocationRegion /> |  |
| DLocationCity | First Event | <remv1:DLocationCity /> |  |
| SEntity | First Event | <remv1:SEntity /> |  |
| DEntity | First Event | <remv1:DEntity /> |  |
| SZone | First Event | <remv1:SZone /> |  |
| DZone | First Event | <remv1:DZone /> |  |
| Message | First Event | <remv1:Message /> |  |
| Protocol Number | First Event | <remv1:ProtocolNumber /> |  |

##### Date and Time Formatting

Date and Time Formatting options are supplied for both AlarmDate and NormalMsgDate. This setting allows for the specification of two optional attributes “TimeZone” and “FormattingString”.

TimeZone optionally allows the user to configure a specific time zone to convert the date and time output. Inside the configuration the time zone is specified by the Windows ID used for the time zone (This value can be looked up within the Windows Registry under “HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Time Zone”). If no value is provided, the time zone of the machine creating Remediation Actions is used (the ARM).

FormattingString is used to define the format of the date and time information. This allows the user to customize both which parts of the date and time are available to the executing program and what structure this information should have. Omitting this option causes the date and time to be formatted by the default .NET date time format. Options for creating the format string are defined in the following specifications:

[MSDN: Standard DateTime Format Strings](http://msdn.microsoft.com/en-us/library/az4se3k1(v=vs.71).aspx)

[MSDN: Custom DateTime Format Strings](http://msdn.microsoft.com/en-us/library/8kb3ddd4(v=vs.71).aspx)

Since both of these values are optional, there are four permutations TimeFormat can take. But any field accepting a TimeFormat must have exactly one TimeFormat specified.

Default Values for the output format and time zone:

<remv1:TimeFormat />

Specify only the Time Zone, Use the .NET Date Time Format:

<remv1:TimeFormat TimeZone="Eastern Standard Time" />

Specify only the Format, Use the Event Manager’s Time Zone:

<remv1:TimeFormat FormattingString="MMMM dd, yyyy" />

Specify both the Format and the Time Zone:

<remv1:TimeFormat TimeZone="Eastern Standard Time" FormattingString="MMMM dd, yyyy" />

##### Parameter Sort Order

Until this point only command-line arguments with no distinction in their order have been presented. However many executables and scripts place emphasis upon the location of the argument. To accommodate the Actions against scripts and executables where the order of the arguments has significance every Constant and String Parameter must define an order.

Order can be defined as one of three values: Unordered, Positive Numbers, and Negative Numbers.

Unordered values as the name suggests are not guaranteed to exist in any particular order, this is the default Order of both Constant and String Parameters.

Positive Numbers identify values starting immediately after the command. This indicates that a Parameter with Order set to 0 produces an argument that appears first in the argument string, Order=”2” produces the second argument and so on. Finally, negative numbers define arguments in relation to the end of the argument string. This means that a Parameter with Order set to -1 will produce the last argument in the argument string, -2 produces the second to last argument, and so on.

The choice of defining Order depends upon the expectations of the executable or script that the action will call. If the script or executable expects arguments to exist in any particular order or placement, then the action should define the order of those arguments on the Parameter. For example:

Order is insignificant:

<remv1:ConstantParameter Name="a1" Switch="arg1" Order="Unsorted" />

<remv1:ConstantParameter Name="a2" Switch="arg2" Order="Unsorted" />

Can produce either result:

cmd.exe arg1 arg2

cmd.exe arg2 arg1

Order starting after the command:

<remv1:ConstantParameter Name="a1" Switch="arg1" Order="0" />

<remv1:ConstantParameter Name="a2" Switch="arg2" Order="1" />

<remv1:ConstantParameter Name="a3" Switch="arg3" Order="2" />

Produces:

cmd.exe arg1 arg2 arg3

Order starting at the end of the arguments:

<remv1:ConstantParameter Name="a1" Switch="arg1" Order="-1" />

<remv1:ConstantParameter Name="a2" Switch="arg2" Order="-2" />

<remv1:ConstantParameter Name="a3" Switch="arg3" Order="-3" />

Produces:

cmd.exe arg3 arg2 arg1

#### Plug-in/Action Serialization

The Remediation actions take on differing forms during their lifetime through the application. Within the plug-in the actions exist as a configuration file, which defines a plug-in and a collection of actions as XML. Once imported to the database as a registered action, the actions are serialized into binary to keep the size small and are split into one action per row in the database.

Developer Note: The serialization logic (both Binary and XML) is located within the lrautormdn project under the Serialization folder (namespace: SC.lrautormdn.Serialization).

#### Plug-in Versioning

Versioning is only supported for LogRhythm created plug-ins. In order for a plug-in to be a newer version of an existing plug-in it must satisfy a few conditions:

* The Plug-in GUID must be identical
* Every action of the older plug-in must also exist in the newer configuration
* Every action must be unique for the plug-in (no identical actions)
* The plug-in must define the attribute IsLogRhythmPlugin=”true” within the Configuration File

This permits the following changes within a newer plug-in:

* The version number must be larger than the previous versions
* There may be new actions added to the configuration file
* The executable and script contents of the plug-in may change, so long as they still satisfy all the actions that already exist in the plug-in.

Adherence to these rules is critical when ensuring that newer plug-ins will properly replace older plug-ins. If a plug-in needs to be replaced with either a reduced set of available actions or a different configuration of existing actions, this plug-in should be treated as a new, different plug-in. This way any alarm rule configured on the original plug-in can be properly reconfigured on the Alarm Rule and the original plug-in can be phased out manually.

## Automatic Remediation Rules (ARR)

### Registered ARPI

A single registered action will be associated with an Alarm Rule. The configuration of the registered action determines what Plug-in program or script is used to execute the remediation action. The registered action also defines which values can still be configured by the user when defining it on an Alarm Rule (see the Configuration sections on String Parameters and Encrypted Parameters).

### Automatic vs. Managed Remediation

Managed Remediation defines the ability to require approval after the remediation action is generated from an Alarm, but before the action is executed. Each Alarm Rule can be configured for up to three levels of approval before executing. Each level indicates that a unique user must approve an action before it can execute. Users can be configured to approve at different levels, but they may only approve an action once, and they may only approve for one level. If any level runs out of eligible approvers the remediation execution engine will detect this error and automatically cancel the action. If a user at any level denies a remediation action, then the action will not execute. If no approvers are specified in the Alarm Rule configuration, then the remediation action is considered ‘AutoApproved’, meaning that it will execute without waiting for any users to approve it.

Approvers may be configured as either individuals or notification groups, allowing LogRhythm users to leverage existing Notification groups as approvers.

Behind the scenes: The user is not given an option as to what level they are going to approve, just the option to approve or deny. This being the case, the level any particular user approves at is defined by number of users that can approve each level. If a given user can approve at two different levels, then the level with the least available approvers is select as the level the specified user approves. So if a user X is the only individual that may approve level 2, but just one of many users that can approve level 1 then user X will approve level 2. Then another user from level 1 will still be required to approve the action before it will execute.

### Approval Notification (SMTP)

When a remediation action is generated from an alarm, any approver that is configured for that action will be notified via email. Remediation Action Approval notifications will be transmitted using the existing ARM SMTP configuration settings. If configured and email addresses are available, an email will be generated for every address of the available approvers.

|  |
| --- |
| **Subject:** LogRhythm Remediation Response Pending Approval (AlarmID: 81) |
| **Body:**  AlarmID: 81  Alarm Rule: Test Alarm Rule  Sent To: User, Sample 01  Remediation Action Name: "Sample 1:Sample Action"  Remediation Action Command: "powershell.exe -File listInfo.ps1" |

Emails contain the triggering Alarm ID in the title for quick reference. The contents include the Alarm ID, Alarm Rule Name, the Person’s name, and the remediation action and command to be executed.

## Automatic Remediation Engine (ARE)

Automatic Remediation is executed within the ARM Notification Engine and runs on a separate thread. The function of the ARE is to monitor, queue and execute remediation actions for new Alarms. When a new Alarm is raised, remediation actions and approvals are generated based upon the configuration on the Alarm Rule.

The Alarm Rule Remediation configuration is stored in tables EMDB.AlarmRuleAction and EMDB.AlarmRuleActionApprover. The generated executable action is stored in table Alarms.AutoRmdnAction, with all available approvers (see Managed Remediation) stored in Alarms.AutoRmdnActionApproval.

### Remediation Actions

Remediation Actions are executable operations, which can be pending execution, pending approval, cancelled or executed. These actions store the state of any remediation action that has been generated and can be execute. Remediation Actions are stored in a database table Alarms.AutoRmdnAction.

A Remediation Action consists of the following properties:

|  |  |
| --- | --- |
| Property | Description |
| AutoRmdnActionID | Auto incrementing bigint PK |
| ActionStatus | This is the status of the Remediation Action. Status values are:   * Approve/Deny = Pending approval of user * Pending = Pending approval of someone in addition to the user * Executing = Awaiting execution by the ARM or currently executing * Executed: Success = Action was executed and the return code from the plugin was 0 (success) * Executed: Failure = Action was executed and the return code from the plugin was 1 (failure) * Denied: Action was denied execution * Failed To Execute: LogRhythm failed to execute the Remediation Action. |
| ApprovalLevel1Status,  ApprovalLevel2Status,  ApprovalLevel3Status | These reflect the current state for each approval level, these are update with changes to Alarms.AutoRmdnActionApproval:   * Approved: A User approved execution at this level * Auto Approved: There was no approvers configured for the given level, so it is automatically approved. * Denied: A User denied execution at this level * Denied – No Pending Approvers: No approvals were located for the given level. The engine denied execution. * Unapproved: Waiting for approval |
| AutoRmdnPluginID | The ID of the plugin in the EMDB table |
| Executable Action | A serialized remediation action that is ready to execute. |
| AlarmID | If provided this indicates the Alarm that triggered this remediation action |
| Exit Code | If action status is Executed, this is the exit code provided by the application or script that the plug-in action executed. |
| Approximate Execution Time | This is a close approximation of how long it took to the execute the remediation action. |
| Standard Out | The text output returned to the standard console from the executed plug-in script/application. |
| Error Out | The text output returned to the error console from the executed plug-in script/application. |
| DateUpdated | The date this record was last updated. |

This flowchart shows the progression of status changes that each action will go through depending on configuration:



### Approval Management and Monitoring

The ARE will be responsible for ensuring all required approvals are met prior to an action being performed. This consists of sending notifications to all individuals configured for each approval level and continuous monitoring so that when all approval levels are met, the action can be performed. Approval requests and their status will be stored in the Alarms.AutoRmdnActionApproval. Monitoring also includes a maintenance process that will automatically deny Actions if there is an inability to satisfy an approval level, because there are no unique approvers remaining.

|  |  |
| --- | --- |
| Property | Description |
| AutoRmdnActionApprovalID | Auto incrementing bigint PK |
| AutoRmdnActionID | The remediation action this approval record refers to. |
| Level | Specifies Levels 1,2 & 3 for Managed Approvals |
| Status | This is the current status for the approval, this value is kept in synch with the value on the associated level of AutoRmdnAction Table:   * Approved: A User approved execution at this level * Auto Approved: There was no approvers configured for the given level, so it is automatically approved. * Denied: A User denied execution at this level * Denied – No Pending Approvers: No approvals were located for the given level. The engine denied execution. * Unapproved: Waiting for approval |
| PersonID | The person how can/did approve the action at this level |
| AlarmNfnGroupID | If present, this is the notification group ID this person id record was obtained from |
| StatusResolvedTime | The date this record was resolved |
| DateUpdated | The date this record was last updated. |

#### Approved vs. Denied

A single Denied at any approval level will cancel the Remediation Action. However, a denial must be issued prior to all required approval levels being met. If approvals are received from all specified levels, the action will be executed. Only if a denied is received prior to this will the action be cancelled.

## Automatic Remediation Execution Engine (AREE)

The AREE is responsible for executing approved Remediation Actions. When a remediation action is ready for execution it is placed in the Alarms.AutoRmdnAction table. This can occur either from ARE generating an action from an Alarm, or from the Console generating an action to test. The AREE monitors the fields of any pending action until it meets any approval criteria. At this point, the AREE locates the plug-in in the EMDB.AutoRmdnPlugin table. This plug-in is compared to the plug-ins located within the plug-in deployment directory for a matching GUID and Version. If a match is located, the already deployed plug-in will be used. If no match is located, then the plug-in is deployed to its own directory where the contents are deployed for execution. The plug-in deployment directory defaults to CONFIGPATH\config\AutoRmdn\Plugins where CONFIGPATH is the value populated in the registry for the ARM at install time. This location can be modified from the “AutoRmdnPluginDir” advanced property of the Event Manager.

Once the plug-in is deployed, the remediation action is used to define the command arguments used to execute the action. This action is started as an asynchronous process, so other remediation actions can be processed and execute. Once a program completes, the Alarms.AutoRmdnAction table is updated to reflect the results of the execution. This information includes the program’s exit code, an approximation of the action’s execution time and the first 1024 characters of any text output sent to the Console or Error outputs. Since programs and scripts vary in their communication of success or failure, LogRhythm makes no assumption about whether the program succeeded in its effort, but the exit code, and text output, should aid in identifying this.

An action failure is reported if for some reason the created action could not be executed. This could be because the plug-in failed in deployment, the command executed could not be found or any reason that LogRhythm was unable to successfully execute the action.

Plug-in deployments are removed from the deployment directory if they are not currently in use and are older than 7 days. This property is not configurable by the user.

All logging from the ARE and the AREE is sent to the scarm.log, all logs of levels Error and Warning are also provided for the Windows Event log.

**Database Changes [Schema, Tables, anything related to DB Designs in relation to the feature]**

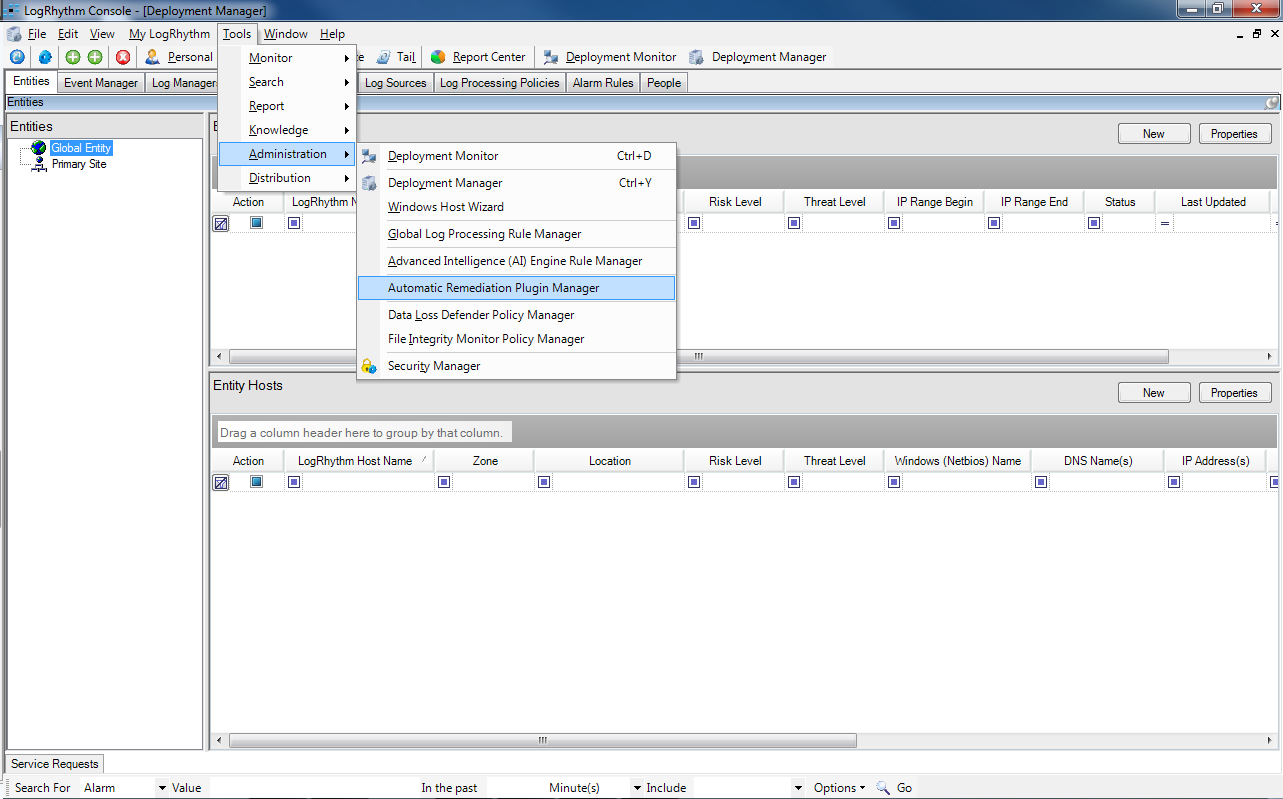
| **DB** | **Change Type** | **Details** |
| --- | --- | --- |
| EMDB | <new AutoRmdn Plugins Table> | Table where the Imported Plugins are are stored for deployment |
| EMDB | <new AutoRmdn Registered Actions Table> | Table where each registered action of a plug-in is stored for use when configuring against an Alarm Rule. |
| EMDB | <new AlarmRule Action> | Table which stores an AutoRmdn Action that is configured for execution when the associated AlarmRule is used while generating an Alarm. |
| EMDB | <new AlarmRule Action Approver> | A single approver (user or notification group) that is configured to against an AlarmRule Action. |
| Alarms | <new AutoRmdn Action> | Table used for Executable AutoRmdn Actions that are generated from an Alarm or a user when testing an action. This contains the current state of the action whether it is still pending execution, executed, or denied. |
| Alarms | <new AutoRmdn Action Approval> | Table used for storing the user approval state of action. This can be a single entry for each approval level when the action is approved or denied, one or more entries when there is an approval pending for a particular level, or zero entries when there are no approvers remaining for a level or the level is automatically approved. |
| EMDB | <new AutoRmdn Approve Action stored procedure> | This stored procedure handles the actions taken when a user chooses to approve or deny a particular AutoRmdn Action. |
| EMDN | <edit LogRhythm\_  EMDB\_  RecordMaintenance stored procedure> | Updated to periodically remove AutoRmdnActions and Approvals when the actions are more than 7 days old. |

Automatic Remediation User Interface

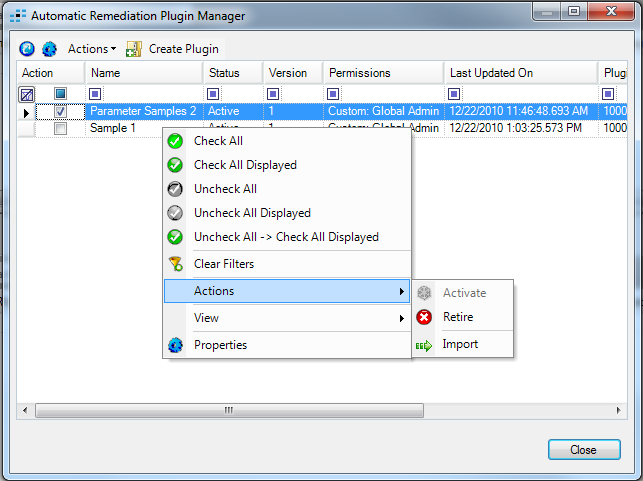
# Prerequisites

The Automatic Remediation Plugin Manager is only available to global administrators.

To display the form, select Deployment Manager, then the Tools menu, Administration, Automatic Remediation Plugin Manager.



# Automatic Remediation Plugin Manager



## Toolbar

### Refresh

The refresh button will reload the form with the latest information from the database.

### Properties

The properties button will be enabled when a plugin row is active. Clicking on it will display the Automatic Remediation Plugin Properties form for that plugin.

### Actions

This is the same actions menu that is displayed in the context menu.

### Create Plugin

The Create Plugin button displays the Create Automatic Remediation Plugin form.

## Columns

### Action

The standard action column is used to check rows for performing batch actions.

### Name

The name of the plugin is defined in the configuration file and displayed here.

### Status

The plugin status can be active or retired.

### Version

The plugin version is defined in the configuration file and displayed here.

### Permissions

This column identifies custom vs. system plugins.

### Last Updated On

Standard column displaying last modified date and time.

### Plugin ID

The database assigned ID for the plugin.

### Plugin GUID

The guid is defined in the plugin configuration file by the plugin creator. It is used to upgrade plugins. Upgrading plugins is only supported for superusers and they can only add actions.

## Context Menu

### Check All

Standard command to check all plugins in the list.

### Check All Displayed

Standard command to check all plugins that are not filtered out or in collapsed groups.

### Uncheck All

Standard command to uncheck all plugins in the list.

### Uncheck All Displayed

Standard command to uncheck all plugins that are not filtered out or in collapsed groups.

### Uncheck All -> Check All Displayed

Standard command to combine two operations.

### Clear Filters

Clears all filters in the grid so all rows are visible.

### Actions

#### Activate

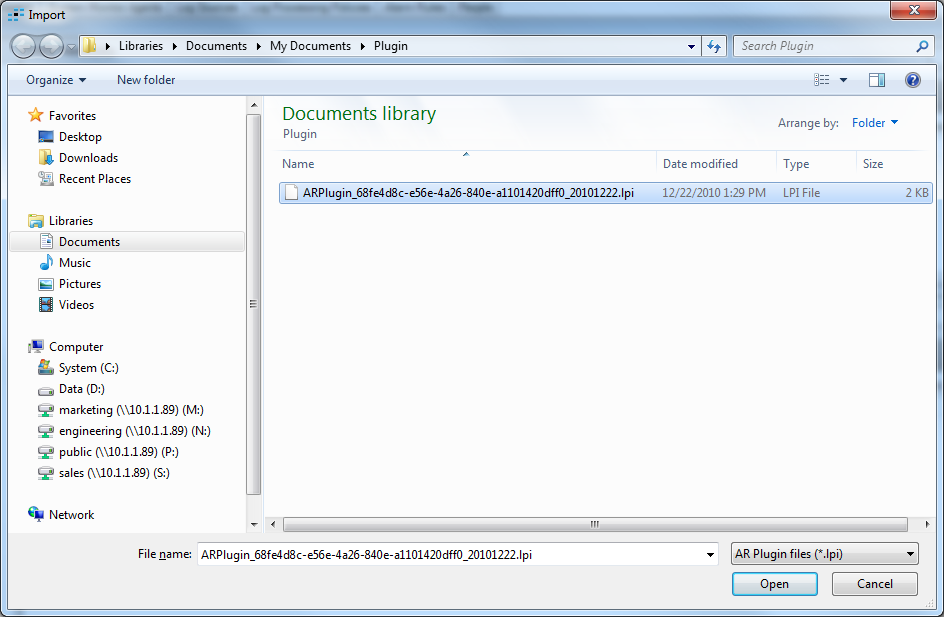
Standard command to undelete a plugin.

#### Retire

Standard command to remove a plugin without physically deleting it.

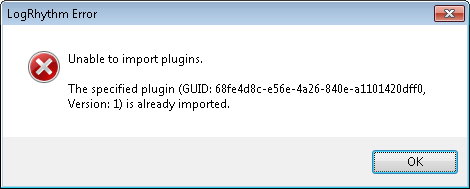
#### Import

Use this to import one or more plugins by selecting plugin package files.



If successful, the plugin will be displayed in the list.

If the plugin has already been imported, then an error message will be displayed.



### View

#### Retired Plugins

Normally rows for retired plugins are not displayed on the grid. This command makes them visible.

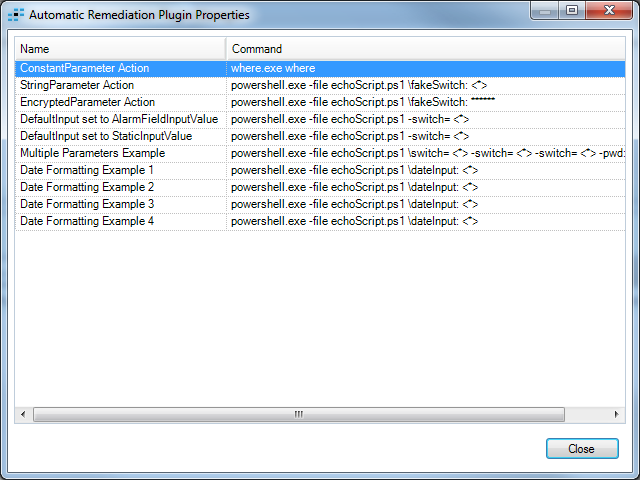
### Properties

Displays the Automatic Remediation Plugin Properties form for the active plugin in the grid.

## Double Click

Double clicking on a row in the grid displays the properties for that plugin.

# Automatic Remediation Plugin Properties



This form lists all the actions in a plugin. It is read-only.

Encrypted parameter values will be displayed as \*\*\*\*\*\*\*\*.

Unconfigured parameters will be displayed as <\*>.

## Columns

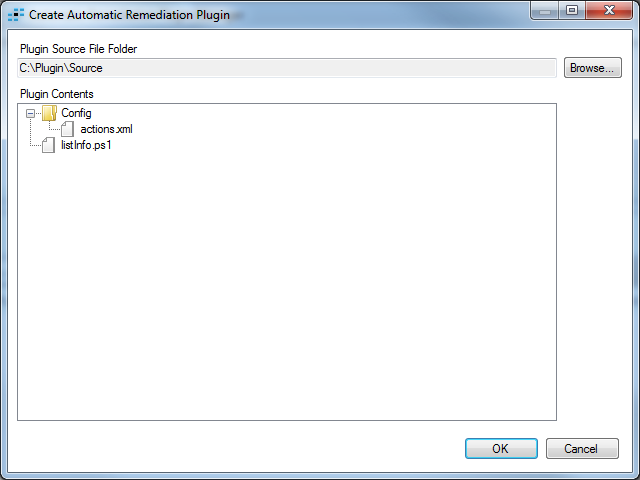
### Name

The name of the action is defined by the plugin creator in the configuration file.

### Command

The command lists the executable the action will run in addition to any arguments.

# Create Automatic Remediation Plugin



The plugin author must create the appropriate files and folders as defined in the spec.

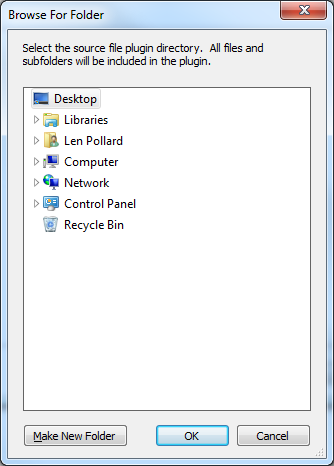
Use this form to browse to the folder and press OK to create the plugin.

## Plugin Folder

This field is read only. Use the browse button to select a folder.

## Browse

Click this button to display the Browse For Folder form. Select the appropriate directory and click OK.

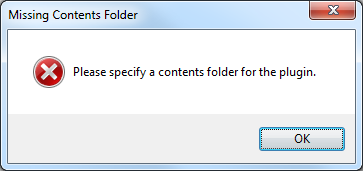


## Plugin Contents

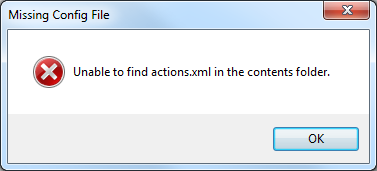
This read only tree lists the folder, its files, and the contents of subfolders.

## OK

When the OK button is clicked, the form is validated. If there are any errors, an error message will be displayed and the form will not be closed.

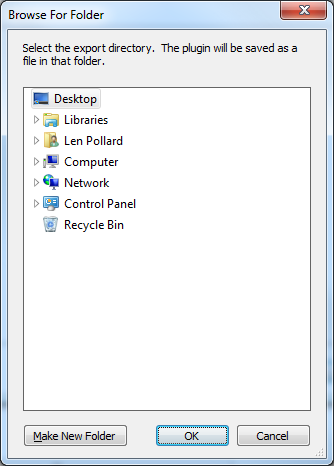


This message is displayed if they press OK without specifying a folder.



This message is displayed if Actions.xml cannot be found in the selected folder or one of its subfolders.

If there are no errors, the Browse For Folder form will be displayed, this time to allow the user to select an export directory.



The exported file will have a system defined name in the format:

ARPlugin\_guid\_yyyymmdd.lpi

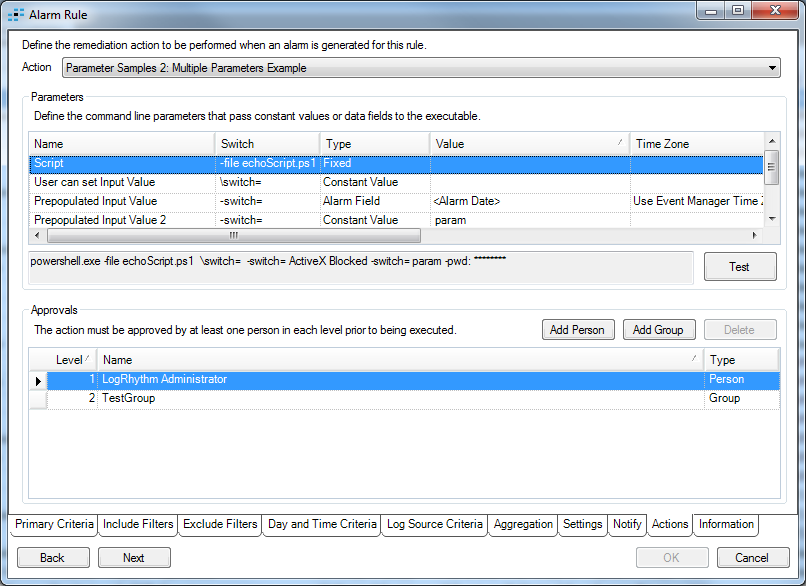
## 

# Alarm Rule Actions

An actions tab has been added to the Alarm Rule properties form and AIE Rule properties form.

Alarm rule actions will be read-only for superusers.

AIE Rule Actions will have their available fields limited to the one group by field. Users should group by the field they want to pass to the action. For example, group by Login for an action that disables a login.



## Action

This drop down control will be populated with all the actions for all the plugins imported in the system. It will use the format “Plugin Name: Action Name”. The value can be set to “None” to specify no remediation action should be performed when the alarm is generated.

## Parameters

The list of parameters are defined by the plugin creator.

### Name

The name of the parameter is read-only.

### Switch

The switch for the parameter is read-only.

### Type

The type can be Fixed, Alarm Field, Constant Value, and Encrypted Value.

The user can change the type between Alarm Field and Constant Value.

Fixed and Encrypted types cannot be modified by the user.

### Value

For Fixed types, the value is read-only and blank.

For Constant Value types, the value is user configurable.

For Alarm Field types, the value can be selected from a list of alarm fields.

For Encrypted types, the value is user configurable.

### Time Zone

Some alarm fields allow the time zone to be selected from a list.

### Time Format

Some alarm fields allow a time format to be specified.

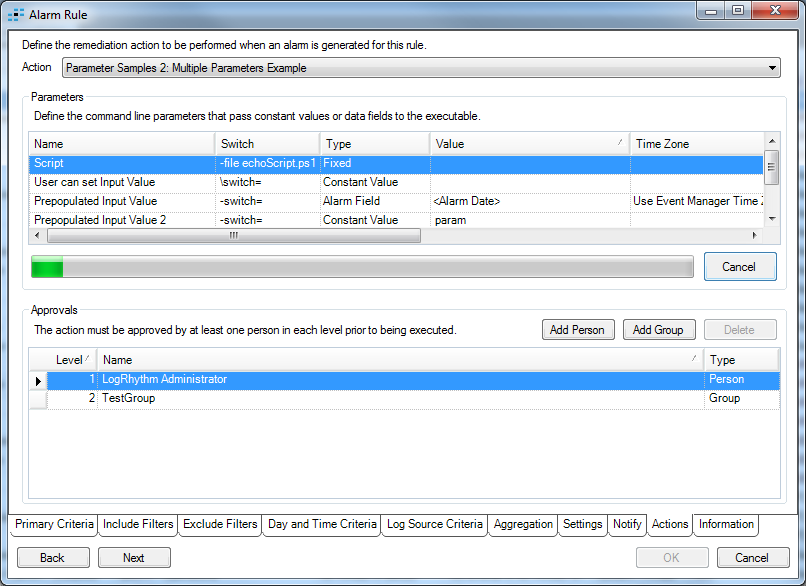
### Command

A read-only sample of the command line that will be executed is displayed here.

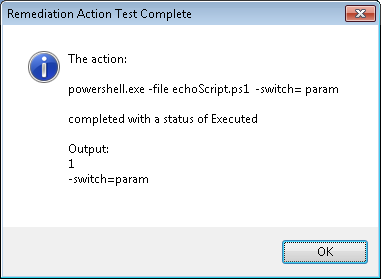
### Test

The Test button can be used to test the action.

The button will change to Cancel and a progress bar will be displayed while the test is running. The button will change back to Test when the test is complete.

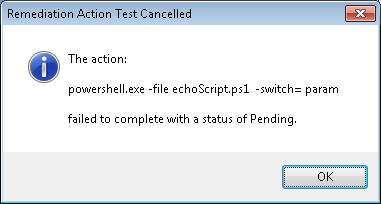


## Test Complete Status



A subset of the standard output and error output will be displayed.

## Test Cancelled Status



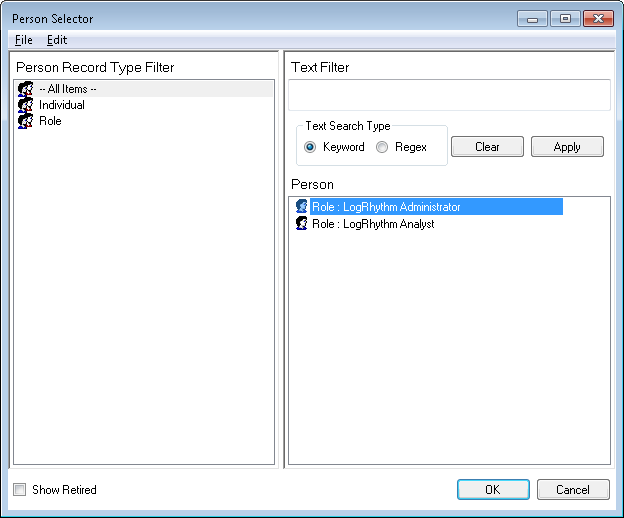
## Approvals

If no approvals are defined then the action will execute when the alarm is generated.

Each approval has a level of 1 to 3. Only one person in each level must approve the action.

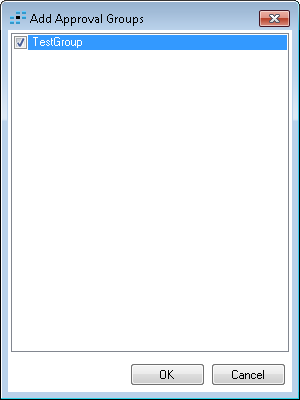
### Add Person

The add person button displays the person selector form.



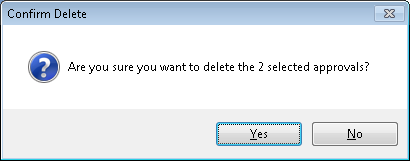
### Add Group

The add group button displays the add group form.



### Delete

The delete button deletes the selected approvers after confirmation.



### Level

The level column has a drop down that can be used to select level 1, 2, or 3.

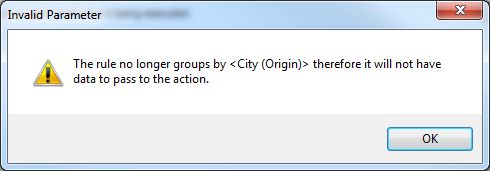
### Name

The name column displays the person or group name.

### Type

The type column displays Person or Group.

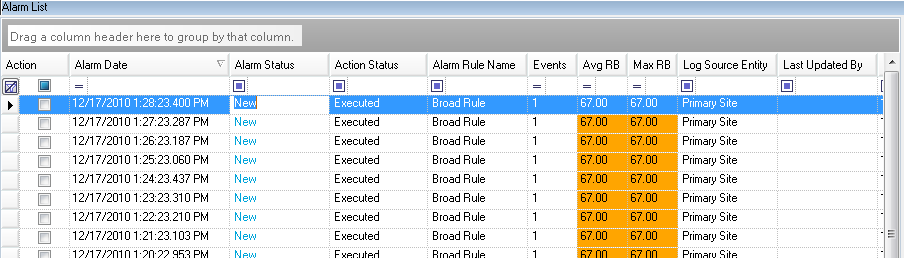
### Warnings



This warning will be displayed for an AIE rule when the user removes a group by field that is used as a parameter.

# Alarm List

The alarm list on the dashboard has a new column: Action Status



Possible values:

Pending: The action has been scheduled.

Executed: The action has been run.

Failed To Execute: LogRhythm was unable to run the action. See the LogRhythm logs for details.

Approval: The action needs to be approved.

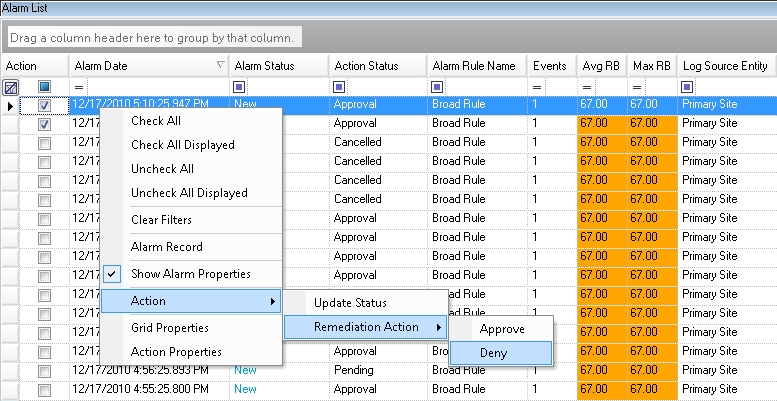
Cancelled: An approver denied the action.

[blank]: An action could not be generated for the alarm. Possibly because an alarm field used as a parameter didn’t have data. See the LogRhythm logs for details.

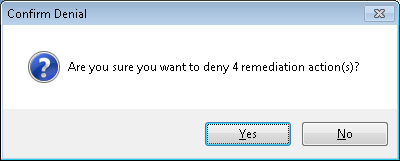
## Context Menu

The alarm list context menu has a new item: Action Properties. This will be enabled when the active alarm has an action.

The Action Menu has Remediation Action -> Approve/Deny which can approve or deny alarm actions in batches.

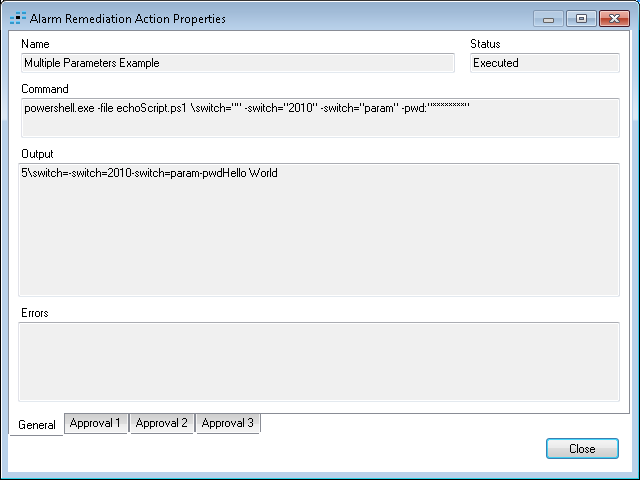


The user will be prompted for confirmation.



Any checked alarms with pending actions where the logged on user is an approver will be approved/denied.

## Action Properties

The action properties form displays details for the action. 

### General Tab

#### Name

The name of the action.

#### Status

The status of the action.

#### Command

This command contains any alarm field values.

#### Output

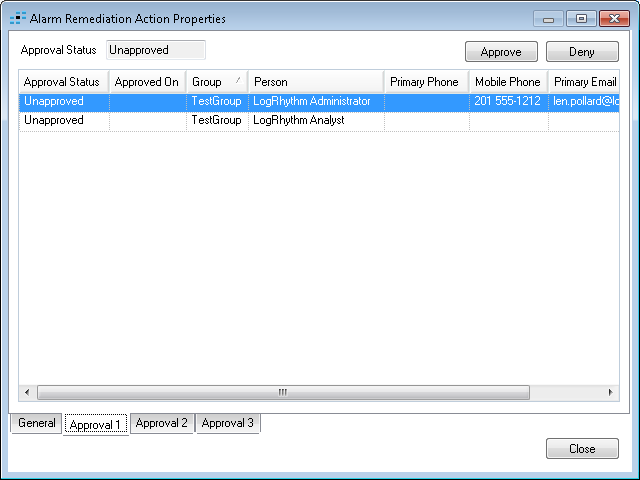
The standard output of the action, if any.

#### Errors

The error output of the action, if any.

## Approval Tabs

There are three approval tabs, one for each level.



### Approval Status

The approval status of the action for the current level.

### Approve Button

Enabled if the logged on person is the approver and the approval status is unapproved and the action status is pending. Click to approve the action.

When the approve button is clicked, the approval might not be applied to this level. All four tabs of the form are updated to reflect the new state of the action. You might need to check the other tabs to see what changed.

### Deny Button

Enabled if the logged on person is the approver and the approval status is unapproved and the action status is pending. Click to deny the action.

### Approval Columns

#### Approval Status

Used to show which person approved or denied the action at the current level.

#### Approved On

Used to show when the approval or denial occurred.

#### Group

If the approval is for a group, then the group name will be displayed here.

#### Person

The name of the person the approval row is for.

#### Primary Phone

Used to contact the approver if necessary.

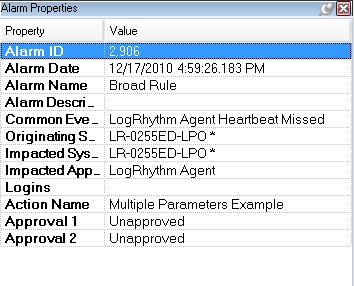
#### Mobile Phone

Used to contact the approver if necessary.

#### Primary Email

Used to contact the approver if necessary.

# Alarm Properties



If the alarm has an action then the action name will be displayed here.

If the action has approvals, then the approval state for each defined level will be displayed here.