CHAPTER 9

Advanced Custom Policy

The Cisco Security Agent (CSA) is an extremely flexible product that has granular policy enforcement capabilities. Included as part of the product installation on the management server is a suite of preconfigured policies that can be deployed to provide immediate protection and control. These policies are a great start and in many cases provide the security required by organizations. In addition, some minor tweaking might be required to allow for approved system use. If you are familiar with the flexibility and applications of the CSA product, you can extend the base capabilities to solve many host security issues in your deployment. In this chapter, you learn about:

- The importance and basics of tuning CSA
- Rule capabilities
- Importance and usage of state sets
- Using dynamic application classes
- Basic forensics

Why Write Custom Policies?

There are several reasons for adding to or changing the default policies that ship with the Cisco Security Agent Management Console (CSA MC). The most common and simplest reason for change occurs during the normal tuning process. The second most common reason for change involves writing custom application control policies to better secure your system. The final reason to change policy is to perform forensic data gathering across the deployment.

The Normal Tuning Process

The normal tuning process occurs during every CSA deployment and continues after deployment when software and patches are added to your systems. These custom policies are often called exception rules, which are rules the administrator creates to allow normal system and application interaction to occur. Often, this also includes changing rules that query the user into straight allow rules that require no interaction. This means you not only tune the policy to allow specific use but also streamline and simplify the user interaction

with the agent, so it does not become a nuisance. If the product becomes too cumbersome for users, they tend to attempt to circumvent the security measure, which would completely go against your goals.

The following are a few reasons to create exception rules:

- **Installers**—You likely have a standard process for installing software in your environment, such as using login scripts and software deployment tools. It is important to allow these processes to maintain your systems unimpeded without user interaction and without weakening the security of your endpoint.
- **Application memory usage**—Many poorly coded applications (or cleverly coded, depending on your frame of reference) might attempt normal data or stack memory access or even attempt to access memory used by another process. You might need to allow these applications to perform this action for them to function correctly.
- **Code injection**—Some applications attempt to insert themselves or DLLs into other processes as part of normal usage.
- **Network access**—You often need to tune systems to allow inbound and outbound access to services on workstations and servers. This can include remote control applications and other network services, such as FTP, TFTP, TELNET, SSH, and HTTP.

Custom Application Control Policies

In addition to creating exception rules for your policy, you also need to craft additional policies that control how other applications are used in your network. Many of the policies written in CSA that control applications are a direct result of your written security policies and acceptable use documents that the users acknowledge. CSA allows you to take the verbiage in these documents and place actual enforcement controls on the systems rather than hoping that your users follow the rules.

Examples of reasons you might write custom application control policies include:

- **Preventing or controling certain application usage**—Your organization might want to prevent or control specific applications, such as P2P files sharing applications, instant messengers, e-mail applications, and remote control products.
- **Limiting system network exposure**—You can institute policies that control which services are available remotely when you connect to the corporate network rather than at a remote, untrusted location. Examples of such connections include a user's ISP connection, a wireless hotspot, or a hotel network.
- **Administrative policies**—You can create policies that limit which users and systems can access administrative tools and also provide higher levels of access to administrative users (or any other users or groups necessary).

- **Application installation policies**—You can create policies that allow CSA to permit mass deployment products to install software unimpeded (examples of mass deployment products include those available from BigFix, Microsoft, and Altiris). Other manual installs can either interactively prompt the user or be denied completely.

Forensic Data Gathering

Because CSA continually monitors system interaction on endpoints in your environment, you might want to leverage this product to report certain interactions you find interesting. By creating a specific set of rules that monitor interaction, you can create a "Honey Pot" policy that when deployed reports interaction of specific processes for you to acknowledge. Monitoring system interaction or at least specific interaction can provide an early warning system that can alert you to suspicious activity before it becomes a real security issue.

Preparing for the CSA Tuning Process

The CSA tuning process is more an art than a science, and you might need some practice before you become efficient. Understanding the points discussed in this section will make you more effective. These include understanding the components of CSA Policy, knowing what protection each rule type provides, and understanding how to use advanced components such as state sets and dynamic application classes. The following sections explain those components and provide a process to follow when you tune or create a custom policy.

NOTE This chapter reviews rather than thoroughly describes each component. For more detailed information, refer to the Cisco Systems website and the Cisco Press book *Cisco Security Agent*. Additionally, this chapter focuses on Windows components to illustrate our points.

Understanding Rule Capabilities

It is imperative that you know the protection provided by each rule type, so that you can quickly write rules without using the Tuning Wizard or researching endlessly. The following is a list of rules most commonly used when tuning. The list is ordered by frequency of use in tuning. You should memorize these components to save time when tuning an environment because memorization greatly simplifies your workload.

- **System API Control**—This rule provides many types of protection to hosts and is by far the most common rule tuned in any deployment. This rule applies to processes as

defined by the associated application class and can provide the following common protections or exclusions:

- **Inject code into other applications**—To function, some applications need to insert themselves or DLL's into other applications. This type of injection can be malicious, however, because viruses often attempt to inject their DLL into a privileged process to gain administrative rights to the system. Be certain that this process is normal before you tune!
- **Write memory owned by other applications**—Occasionally applications attempt to use another application's memory space. This is somewhat uncommon but has been seen in off-the-shelf software.
- **Access systems functions from code executing in data or stack space**—Although this is a common buffer overflow action and should be treated with care, some applications do this to check their licensing. Verify that this is repeatable and normal through active testing or by confirming with the vendor, then tune appropriately. You can tune this rule granularly through the use of pattern matching, which the Tuning Wizard commonly performs.
- **Trap keystrokes**—Some software attempts to capture keystrokes as part of normal behavior. Verify this is not malicious before proceeding.
- **Monitor media devices**—You can control which devices in your system control or access media devices, such as video cameras and microphones.

- **Application Control**—This rule provides the ability to control whether an application is allowed to run. It also controls what applications can start other applications. It is an important rule type, especially when combined with dynamic application classes, which you see later in the chapter in examples of advanced custom policies.
- **File Access Control**—This rule controls which applications are allowed to read and write files and create directories.
- **Network Access Control**—This rule controls how a process is allowed to initiate, terminate, or listen for network connections.

Although you will become familiar with several other rule types through daily use of the product, you should completely understand the rule types in the previous list before beginning to tune and create customer policies in your own environment.

Discovering State Sets

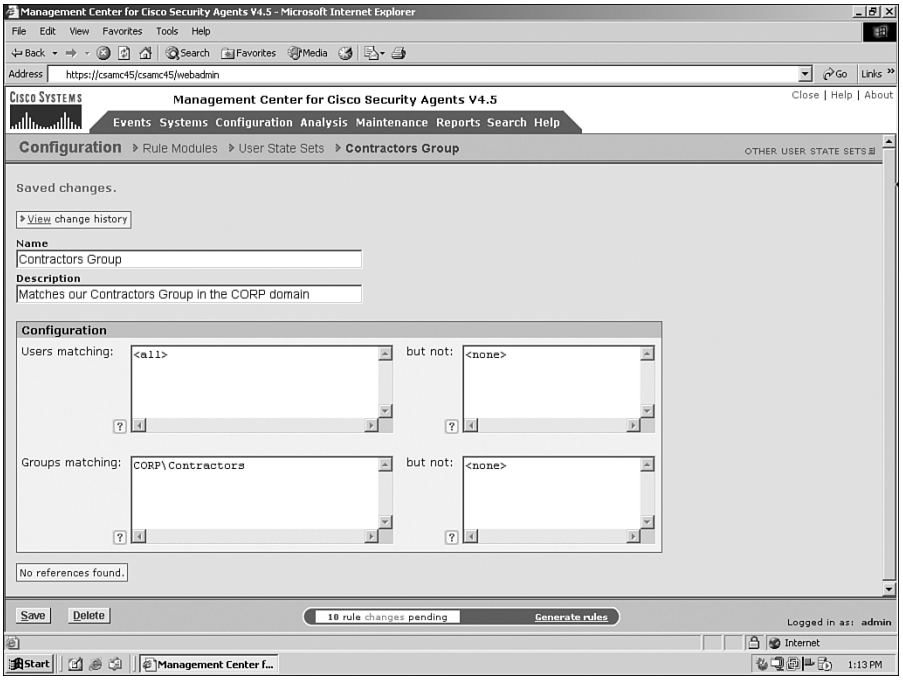
State sets provide a level of granularity and control not provided by many other Host Intrusion Prevention System (HIPS) products. Become familiar with two types of state sets: user and system state sets. These sets provide mechanisms that enable a CSA administrator to deploy policy to endpoints that are enforced only when a specific environment is encountered, such as the administrator logging into the system or a specific IP address used by the computer.

User-State Sets Overview

User-state sets are matched on an endpoint when specific users or groups are in use on the system. You can both define as many of these sets as you want and use the state sets that are pre-installed with the CSA MC. These objects allow you to enforce policy that is not normally allowed. The following examples illustrate this, and a sample User-State Set configuration screen is shown in Figure 9-1:

- **Administrative access to manual installations**—Although your average user might not be able to install software locally, you could allow the administrator to log in and perform the installation. The state set would identify this user account and allow the installation by applying specific allow rules to the system temporarily while the administrator is logged into the system.
- **Remote access to the registry**—You might use management tools to set registry settings remotely that CSA would normally prevent. You could use a user-state set that matches a specific account or group used to authenticate to the local system and override the preventative policies.
- **Administrative CSA control**—You might define a rule module that allows the CSA to be viewed or stopped only when the matching state set is active.

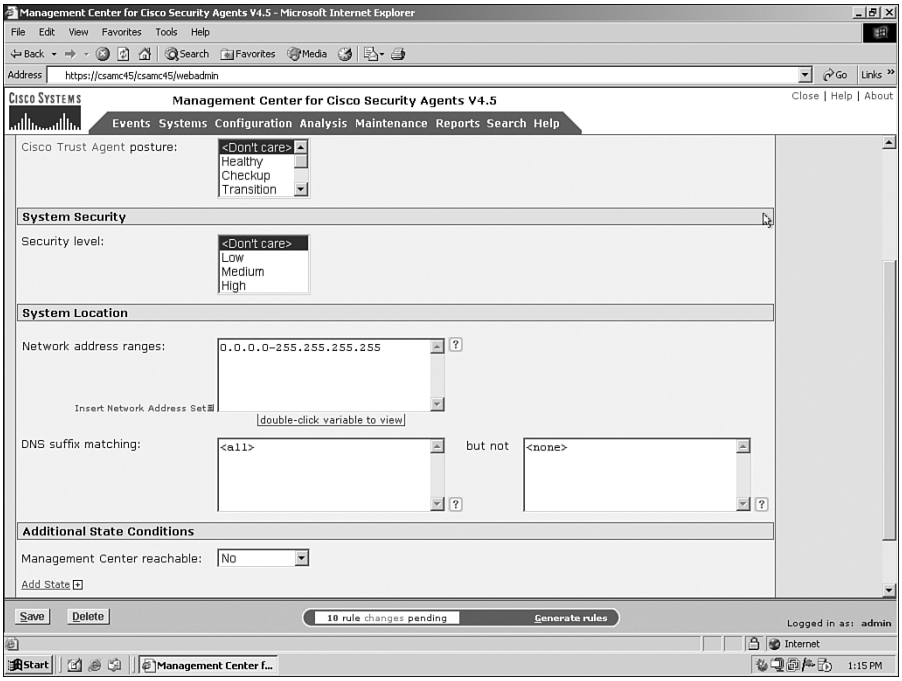
Figure 9-1 *User-State Set Configuration*



System State Sets Overview

System state sets are matched on an endpoint when various criteria are matched. There are several reasons to build a system state set, such as identifying when a system is on or off a specific subnet, when the CSA MC is reachable, when an installation is currently in progress, or when your Network Admission Control (NAC) posture token is changed or set among others. The following is a list of common settings that can be used alone or in conjunction to match a specific state and a sample image of what the System State Set configuration screen looks like in Figure 9-2.

Figure 9-2 *System State Set Configuration Options*



- **Cisco Trust Agent posture for NAC**—You can define a set that matches the various posture settings that NAC provides, such as Healthy, Quarantine, and Infected. You might decide to enforce different rules when the state matches, such as preventing Outlook from opening attachments when NAC has determined the system is infected.

- **Security level**—If you allow users to see the CSA in the system trays of their computers, you can also allow them to use the security-level selector that allows them to change the setting to Off, Low, Medium, or High. These settings can enforce different policies as defined by the CSA administrator.
- **Network address ranges**—This identifies the network to which the user is currently attached.
- **DNS suffix matching**—This identifies the users' DNS suffix, such as ServiceProvider.net, Company.com, and VPN.Company.com.
- **Management Center reachable**—This matches if the agent determines that the CSA MC can be contacted or not. This is a good way to know if the user is currently connected to your network or not connected. You might use this to enforce restricting inbound connectivity for the system if it is not connected to your network.
- **Installation process detected**—This matches when a process is placed into the **<*Installation Applications>** special dynamic application class. This state allows the alteration of the current running policy, so that the installation can continue without too many user-required query responses, if any at all.

Using the combinations of the variables that create system state sets is powerful when building custom policies for your environment. You should try to use these objects when creating policy to ensure granular security policy enforcement as an alternative to creating a policy that is too loose and allows negative actions to occur for all system states. An example of using multiple variables for a system state is determining if a user is VPN-connected. You could match on the system IP address, the DNS suffix, and the CSA MC reachable parameters to determine that a system is connected to authorized VPN concentrators. After this state matches, you can alter the system security policy to allow or deny system functions, such as file transfers or remote control functions if necessary.

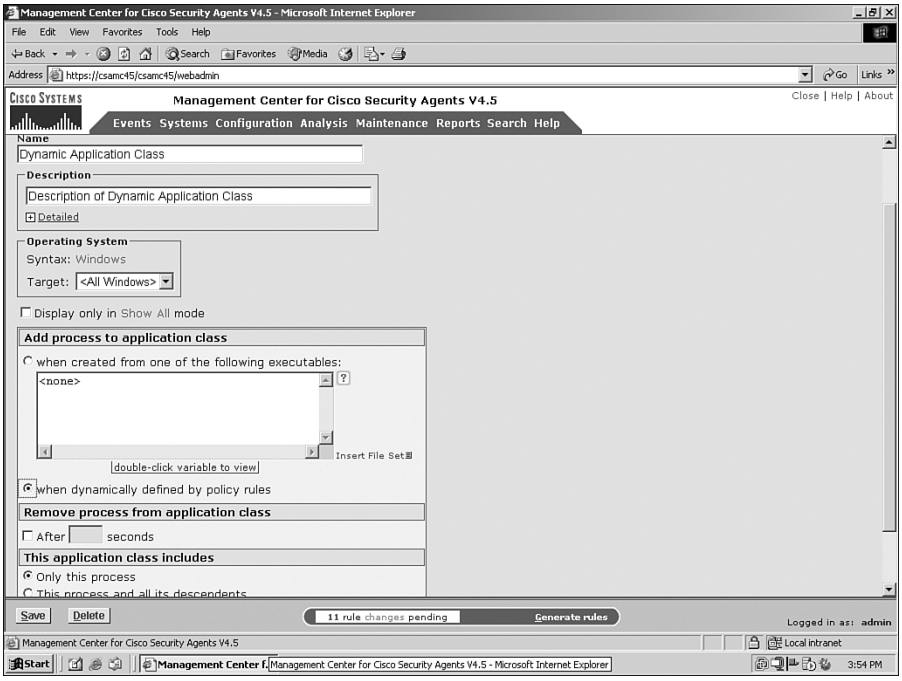
Discovering Dynamic Application Classes

Just as state sets can provide great distinction in levels of policy enforced, dynamic application classes can also provide granular policy manipulation. If you use the dynamic application classes effectively and efficiently, you can simplify the amount of work you need to perform and the number of rules you need to create when tuning processes. In addition to simplifying the number of rules required to maintain your environment, dynamic application classes can provide much stronger security to the endpoint. The following examples describe some common uses of dynamic application classes and Figure 9-3 shows the configuration of a dynamic application class.

- **Telnet applications**—You can automatically add processes to this class when they attempt to access remote IP addresses over TCP/23.

- **Limit executable actions after accessing a protected file**—You could place processes in a special class after they read or write to a specific folder. You could then limit the capabilities of this process to ensure it cannot transmit files or perform other actions.

Figure 9-3 *Dynamic Application Class Configuration*



Best Practices for Tuning

There are many ways to tune a policy, and often there are multiple variations of policies that can accomplish the same tasks. It is often stated that there is no wrong way to tune, but there are definitely some advanced issues you should consider before choosing to tune any rule. Some of the issues to consider include: ease of migration of consistent policies among multiple environments (development and production), ease of transition during CSA MC software upgrades, and the flexibility and strength of the policy.

Understanding Importing and Upgrading

When you design your policy and make changes to the default policy included with the CSA MC, it is important to understand how any changes you make can impact the amount of effort it takes to exchange policies between your production and development environments and also when you upgrade minor or major revisions of the CSA product.

NOTE Many corporations use multiple environments to control their testing and implementation processes and change control impacts. It is not unusual to see this between two and four environments such as: testing, development, systems integration, and production.

When you import objects you have exported from another CSA MC (or a previous export from this CSA MC), you should understand which items are duplicated and renamed and also which items replace the original. Additionally, you should understand that part of every CSA software upgrade, such as moving from CSA v4.5.1.628 to CSA v4.5.1.639, also includes an import process as part of the upgrade.

During software upgrades on the CSA MC, the imports compare each individual imported objects against the current objects to see if there is a match by name. If there is a match, the system determines if the object is an exact match or not. If it is an exact match, the new object replaces the old object and the old one is removed. This means that any policy that uses this object now includes the new object version automatically. If the object is not an exact match and some of the parameters have changed with the upgrade, the new object is imported and displays the new version number, but it does not automatically replace the object in the current policy. You need to perform compares on the new and old object to see what changed in the newly upgraded object and determine if you would like to incorporate the changes.

During an import of an object that is not part of an upgrade procedure, the objects are also compared. If the object name already exists, the system creates the new object being imported but appends to the name to differentiate the newly imported object. The appended name contains an underscore character followed by a portion of the name of the import process you created to import this object, such as _import-name. You need to compare and apply these new objects as necessary using a manual process. Of course, if the new imported object does not match any existing objects, it simply adds the object to the system without changing the name of the object.

You can see that the previous two types of imports can cause you to perform manual tasks upon completion to apply the policy you want and to clean up the post upgrade and import environments. This can be a tedious task. Ensuring that you make as few changes as possible that cause the post-import tasks to grow in number greatly simplifies your job as an administrator. For this reason, it is important that you think about the objects you edit

before you make the changes. At first, it might seem like a better idea to edit the settings in default objects, so that you do not have to create an additional rule to add the functionality you are attempting to add. However, if you do this you actually ensure that any imported or upgraded policy does not match and results in duplication that requires manual cleanup. You should always attempt to leave the default objects simple and unchanged if possible. This is not always the case because there are exceptions to the rule, but if you attempt to make this part of your decision-making process, you will have much simpler administrative tasks in the future.

Variable and Application Class Usage

When creating policy, many types of objects are available to you. Often, because many of the fields available to the administrator allow literal values to be entered along with variables, the administrator enters the value into the fields rather than creating a new variable (such as File Set, Network Set, and so on) or application class. It is recommended that you attempt to create variables and classes as often as possible to allow your future policy to deploy more rapidly. Any object that can be reused later in the software life-cycle simplifies your policy development and also ensures consistency among multiple administrators.

Sample Custom Policies

As with most events in life, seeing is believing. You need to be able to use all the CSA MC policy objects effectively. This section illustrates a few examples of how to build custom policies to assist in constructing your basic skills in this art.

NOTE The sample policies created in this section might need additional rules and components to be completely effective in your own environment. The following examples help illustrate the processes involved.

State-Based Policies

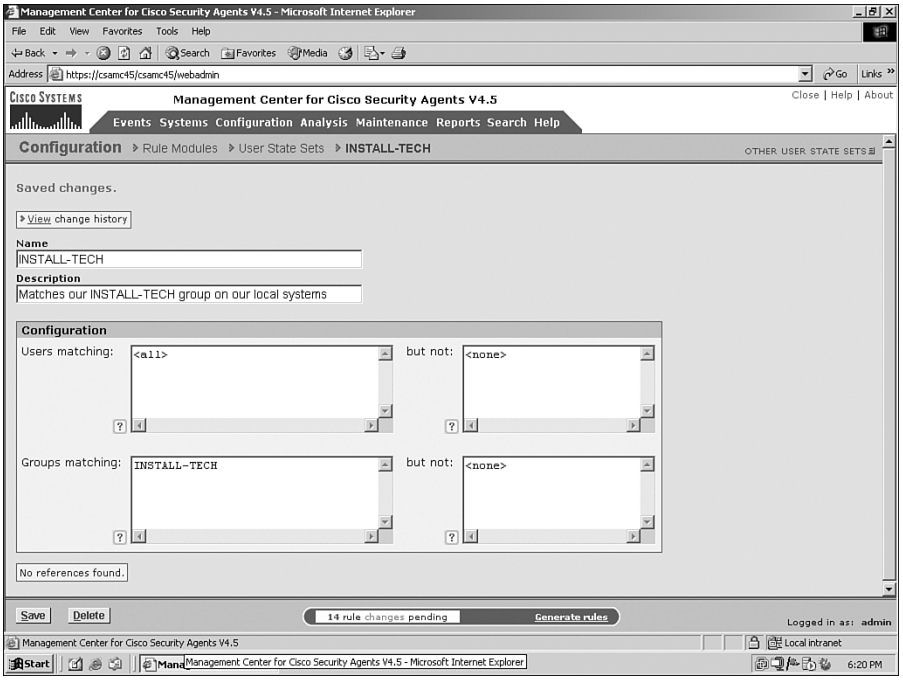
As discussed earlier, state-based policies can be powerful. Using states can be an effective way to lighten policy enforcement on a single machine temporarily without completely degrading the entire deployments security permanently.

Install Technician Agent Control

Often, you encounter the need to allow a local technician of a system to perform actions that would not normally be allowed to the system user. This could be a technician's manual installation of a software package or hardware driver. To accomplish this, you can either use a state-based set or place the system in a special group or test mode so that the installation can be performed. The problem with the last two options is that the CSA administrator would need to be involved in every daily task. In addition, it's possible that the security is completely removed in test mode rather than just slightly degraded and controlled. In this example, we use the following procedure to start to configure the objects.

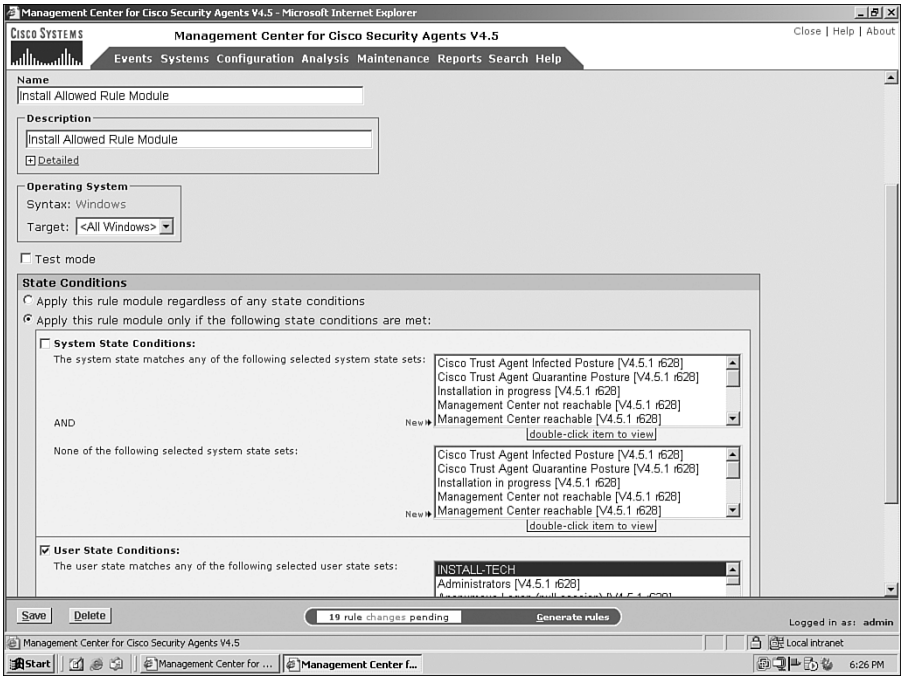
Step 1 Create a user-based state set named **INSTALL-TECH** that matches a local group with the same name. This is effective only if you have a group called INSTALL-TECH on your system and have a user in that group perform the installation. This state set is displayed in Figure 9-4.

Figure 9-4 *INSTALL-TECH State Set*



Step 2 Create a policy named **Install Allowed Policy** and also a rule module named **Install Allowed Rule Module**. The rule module should be enforced only when the INSTALL-TECH state set matches. The configuration for the rule module can be seen in Figure 9-5. The rule module should be associated with the policy.

Figure 9-5 *Install Allowed Rule Module with State Set*

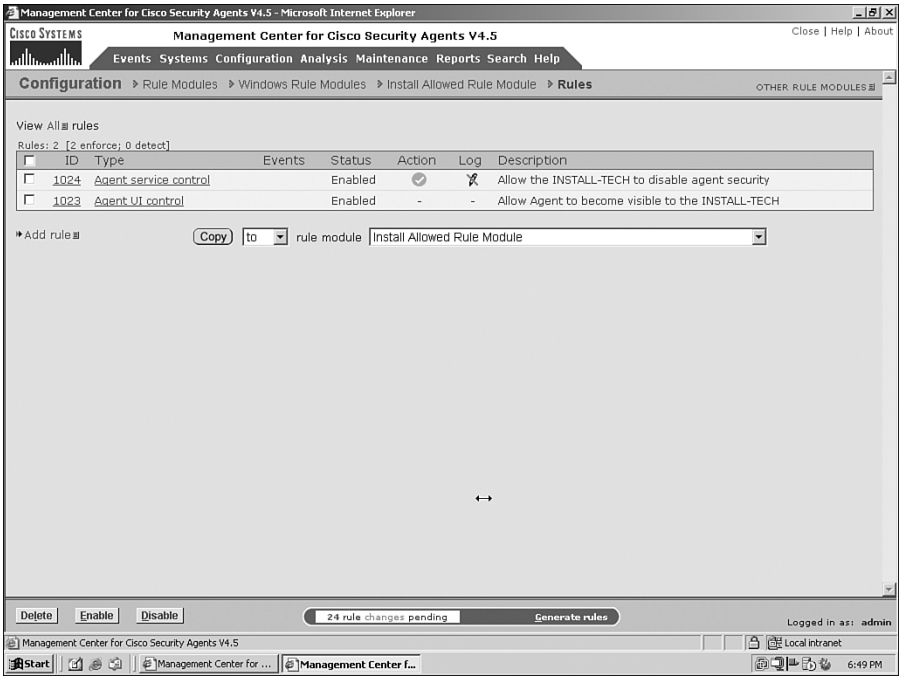


Step 3 Insert the following rules in the rule module. See these rules in Figure 9-6.

- Agent UI control—Allows the agent to become visible to the install technician.
- Agent service control—Allows the agent service to be stopped by the install technician.

Step 4 Attach the policy to groups as necessary.

Figure 9-6 *Add Necessary Rules to the Rule Module*



At this point, an install technician should be able to log in on any system that carries the policy and install software. They can receive query messages and stop the agent service when necessary.

Remote Registry Access

It is not unusual for certain systems in your environment to attempt registry access to workstations for various purposes. To allow this access, yet not open up remote registry access to all systems, you need to use a user-state set. Follow these steps to create the policy required to accomplish this task.

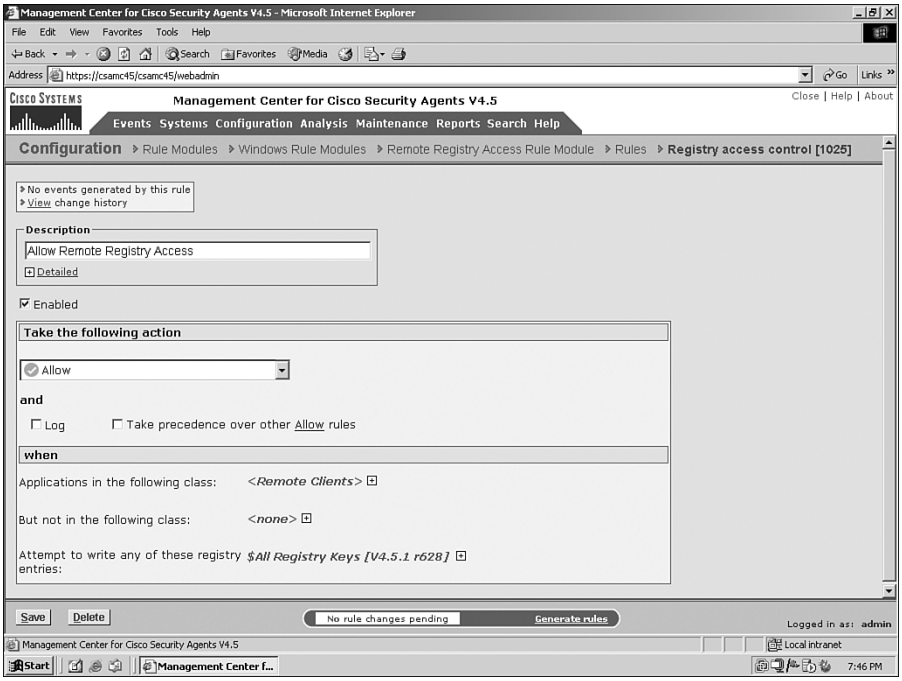
Step 5 Create a user-based state set named **MGMT** that ties to a group that the user who will attempt the remote access is a member. When a user who is a member of this group authenticates to the remote system, the state set will match and your rules will temporarily apply to the system.

Step 6 Create a policy named **Remote Registry Access Policy** and a rule module named **Remote Registry Access Rule Module** that you can associate to the policy. This rule module should be enforced only when your state set matches on the system.

Step 7 As shown in Figure 9-7, add a Registry Access Control rule to the rule module that allows **<Remote Clients>** to access all registry keys.

Step 8 Apply this policy to the correct groups.

Figure 9-7 *Remote Registry Access Rule*



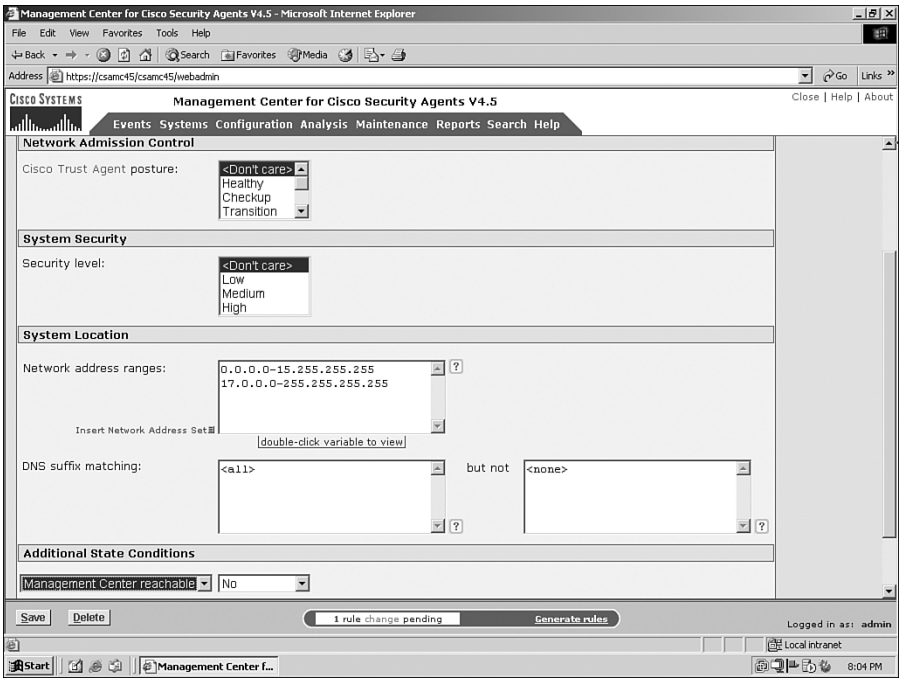
Remember, if you apply this correctly, you allow access to the registry remotely only after a successful authentication of a specific user or group member. This is much more secure than allowing all access to the registry at all times.

Securing the System When Away from Home

When systems connect to your network, corporate firewalls, intrusion detection systems (IDS), intrusion prevention systems (IPS), and other security devices protect them. When they disconnect and travel to remote networks at coffee shops, hotels, or even their home, they lose all that protection. Therefore, it might be desirable to raise the level of network security enforced on these systems when they travel abroad. In many cases, endpoints run many services that listen on the network, such as mass deployment and system management software, remote control packages, file sharing, and web servers. The following example creates a system state set and a policy to help you lock down your systems when they leave your premises.

Step 1 Create a system-based state set named **OFF-NET** that matches IP addresses that are not part of your address space. Also, be certain that the CSA MC is not reachable as shown in Figure 9-8. When a system does not have an IP address you own and also cannot reach the CSA MC server, you can assume that the system is not local.

Figure 9-8 *OFF-NET Systems Set*



Step 2 Create a policy named **OFF-NET Protection Policy** and a rule module named **OFF-NET Protection Rule Module** that you can associate to the policy. Enforce this rule module only when your OFF-NET state set has matched on the system.

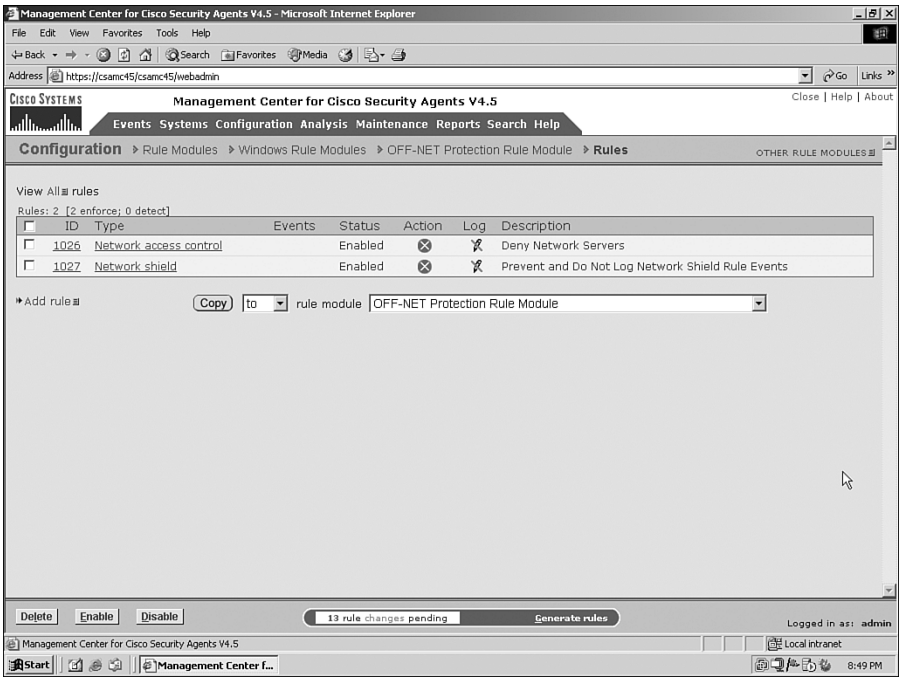
Step 3 Add the following rules as displayed in Figure 9-9:

— Add an NAC rule to the rule module that denies all applications from acting as a server on all TCP and UDP ports. This should be enforced only when you are not connected to the corporate network.

— Add a Network shield rule that prevents all malicious packets and various scanning mechanism but does not log the messages. Your systems are guaranteed to be scanned and see some of the worst the Internet can throw at them when they are off your network. It is therefore advised that you do not log these attempts as you have little recourse when the host is miles away from any protection you can provide. Additionally, you are not likely to receive these messages until the host next connects via remote access or locally, which most likely guarantees you would be too late to react.

Step 4 Apply this policy to the correct groups.

Figure 9-9 *OFF-NET Rules*



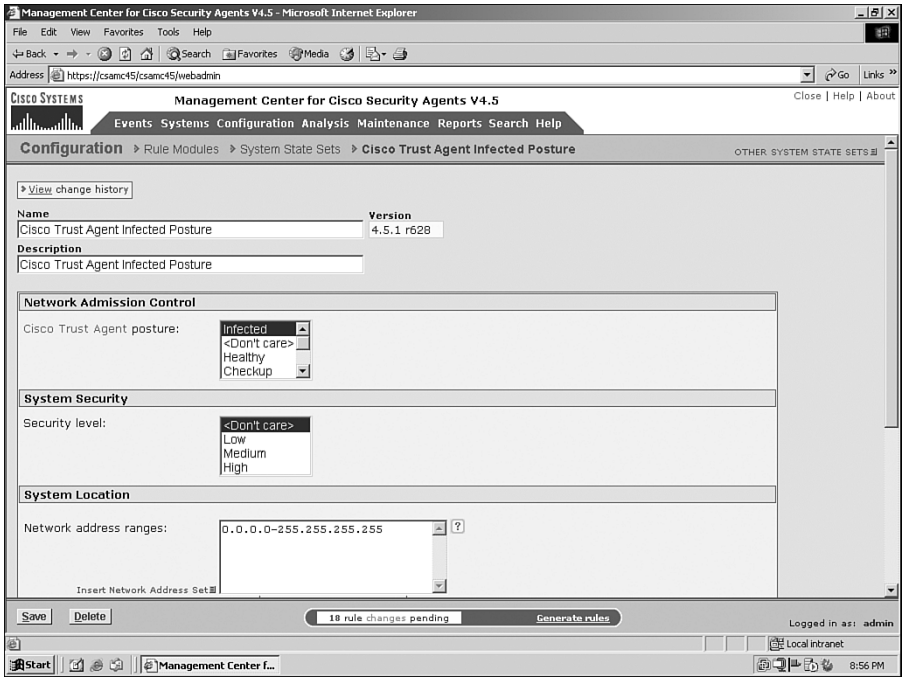
You can add any other rules that would control the system when it is not attached to your network. You might decide that no client connections should be made out of the system except virtual private network (VPN) initiation back to the corporate facility for example.

NAC Policy

If your company decides to implement Cisco NAC, you will find it advantageous to use the CSA to complement the solution. You can use the NAC posture token returned by the Cisco Access Control Server (ACS) Policy Server to match a policy and therefore cause an additional policy to become effective. This example uses the pre-defined Cisco Trust Agent Infected Posture system state set to determine when the NAC server deems your system infected. After you have matched that set, you initiate a rule that does not allow e-mail applications to start other applications, such as viewers.

Step 1 Create a policy named **NAC Infected Policy** and a rule module named **NAC Infected Rule Module** that you can associate to the policy. This rule module should be enforced only when the Cisco Trust Agent Infected Posture system state set has matched on the system. View the system state in Figure 9-10.

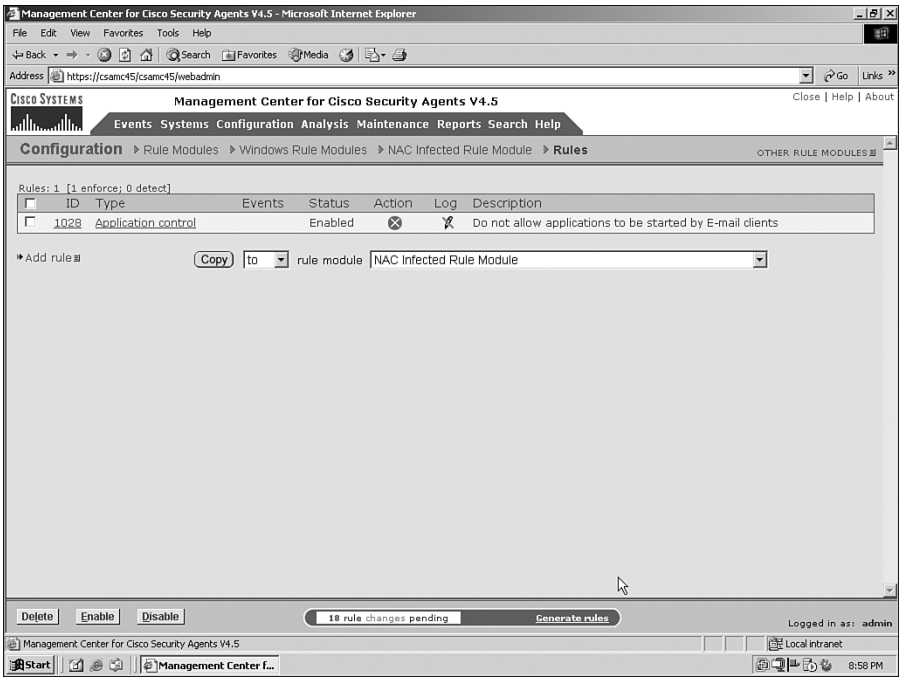
Figure 9-10 *Cisco Trust Agent Infected Posture State Set*



Step 2 Add an application control rule to the rule module that prevents e-mail applications from starting any other application as displayed in Figure 9-11.

Step 3 Apply this policy to the correct groups.

Figure 9-11 *Application Control Rule*



You can add other protections if desired, such as limited network connectivity. As you can see, this type of configuration can alter the endpoint capabilities if the NAC process deemed that you are infected or quarantined. You would then have limited capabilities until you brought the agent-protected system back into a compliant state by using an endpoint management product, such as the BigFix Agent, at which point the system policy can revert to its original policy.

Using Dynamic Application Classes

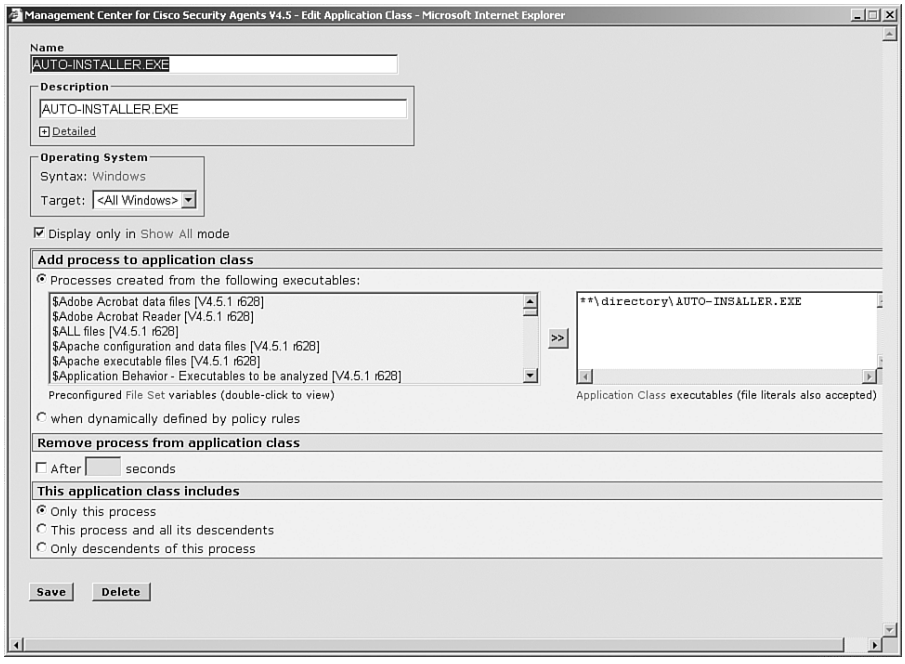
Dynamic application classes are a key component in the CSA architecture. Learning to use these effectively allows you to complete complex tasks using a limited number of rules rather than a great number of static rules. After you understand how to use this type of control, you will use it often and continue to become a more effective and efficient CSA administrator.

An example of dynamic application classes follows. The example tunes an item many corporations use to simplify their environments. Many companies use mass deployment software distribution products to install software, software updates, and patches to their systems automatically. Because the software deployment mechanism is trusted, no CSA administrator wants to tune the policy for every software installation and update. Instead, you can build a rule module that makes use of a dynamic application class to provide the capabilities needed to all the installers that are started by the trusted deployment mechanism.

For this example, assume the software distribution mechanism is named **AUTO-INSTALLER.EXE** and is pre-installed on every system. The following steps walk you through a high-level approach that enables you to create your own policy that allows your enterprise software distribution system to function.

Step 1 Create an application class named **AUTO-INSTALLER.EXE** that includes your application as shown in Figure 9-12.

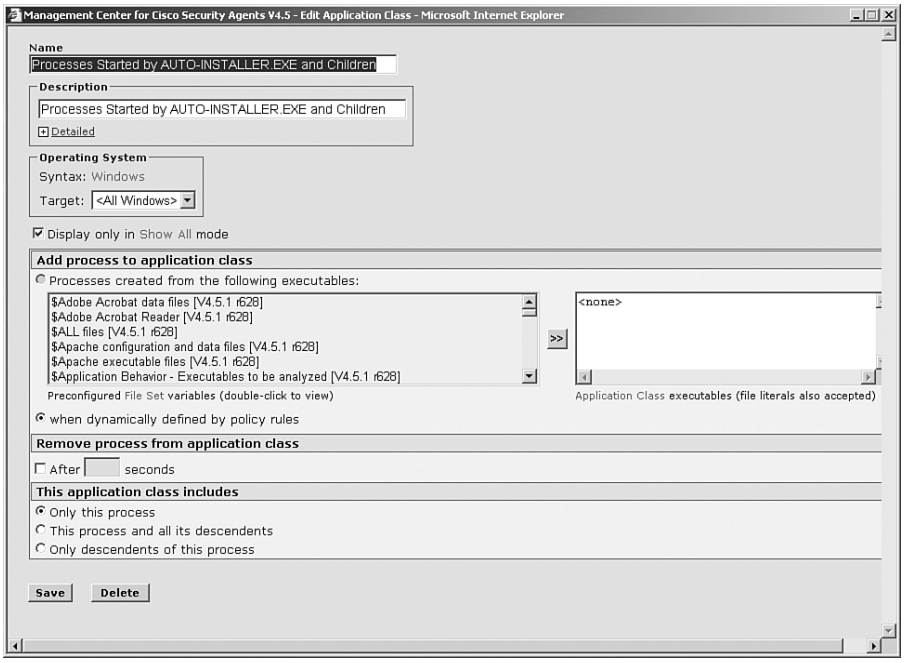
Figure 9-12 *AUTO-INSTALLER.EXE Application Class*



Step 2 Create another application class named **Processes Started by AUTO-INSTALLER.EXE and Children**.

This is a dynamic application class that includes processes started by AUTO-INSTALLER.EXE and their child processes as well. You select only **When Dynamically Defined by Policy Rules** to make this a dynamic application class. Do not select **This process and all its descendants**. You can see the configuration of this in Figure 9-13.

Figure 9-13 *Dynamic Application Class*



Step 3 Create a policy named **Installation Policy** and a rule module named **Installation Rule Module** that you can associate to the policy.

Step 4 Create the following rules in the rule module.

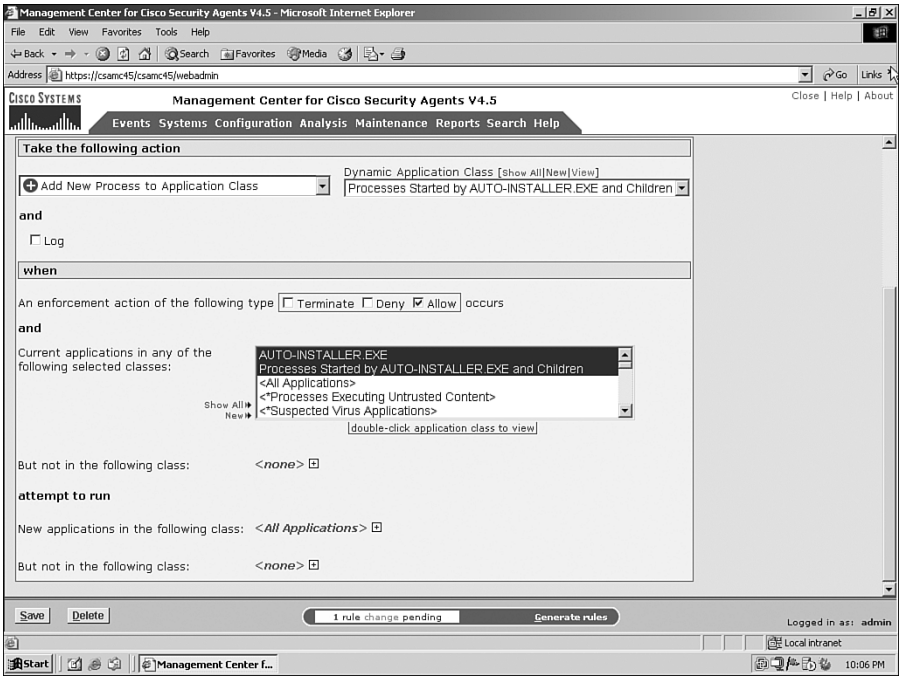
(a) Add an Application Control rule that has an action of **Add New Process to Application Class** and select **Our Dynamic Application Class** from the dropdown list. Select the **Allow** checkbox to list the type of matching action. Select **AUTO-INSTALLER.EXE** and

Processes Started by AUTO-INSTALLER.EXE and Children as the application classes to monitor. Finally, select **All applications** under **attempt to run**. This is shown in Figure 9-14.

This rule can be interpreted as, "When AUTO-INSTALLER.EXE, processes started by AUTO-INSTALLER.EXE, and all subsequent processes start any allowed application, add the new process (as a child) to the dynamic application class."

If you turn on logging for this rule, you can see the process tree of who starts whom. You might not want to do this in a production environment due to the load it can create on the CSA MC, but it can be useful at times.

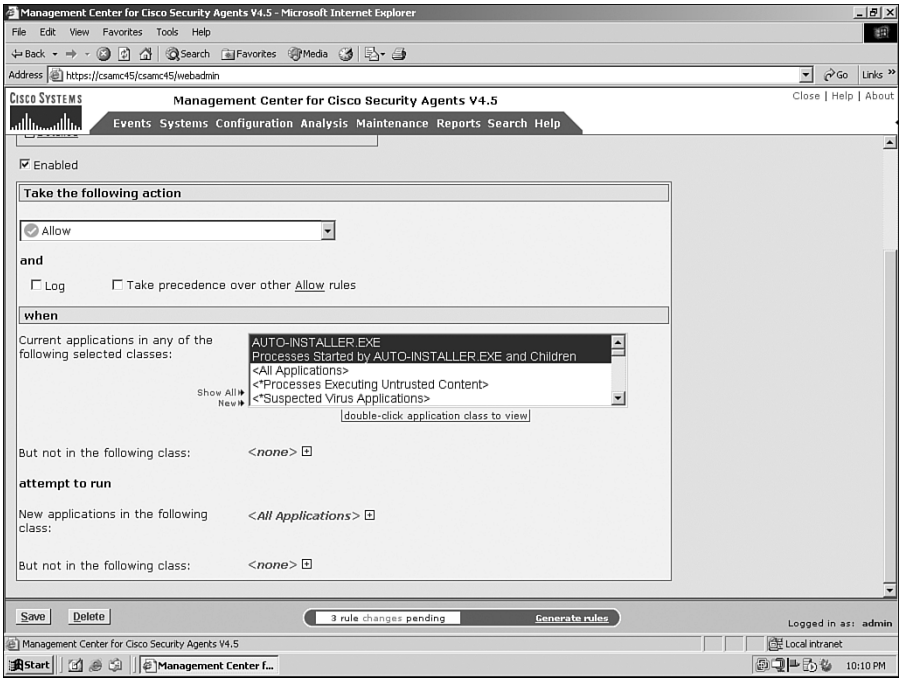
Figure 9-14 *Tagging Child Processes*



(b) Clone the Application Control Rule described previously by selecting the rule from the rule module and pressing the **Copy** button to copy a clone to the same rule module. Edit the new rule and change the action associated to the new rule to **Allow** and also the description as shown in Figure 9-15.

This rule allows the installation program and any software package, patch, or update to start any programs required to complete the installation. This also allows the installer to start the patches and other installers in the first place.

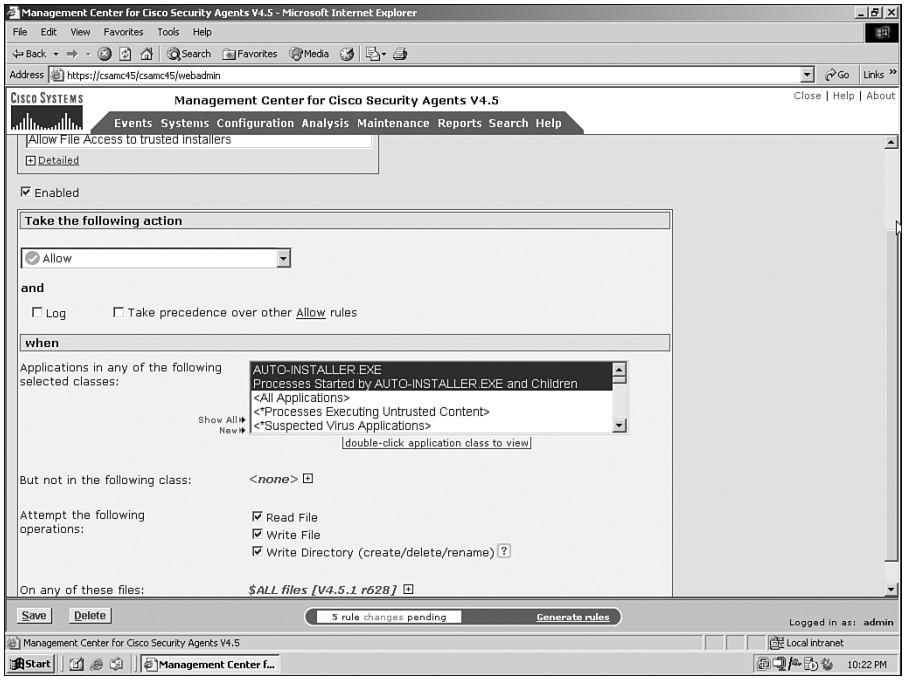
Figure 9-15 *Allow the Installers to Execute Applications*



(c) Add a File Access Control rule to allow AUTO-INSTALLER.EXE and the contents of the dynamic application class to **Read File**, **Write File**, and **Write Directory** on all files. This allows the installer to write to all locations required, which includes writing DLLs and services. It is important to test your installers before allowing your mass distribution system to install the software.

Remember, you assume that anything installed by your installation and distribution program is trusted 100 percent. This rule is displayed in Figure 9-16.

Figure 9-16 *Allow the Installers to Write Files*



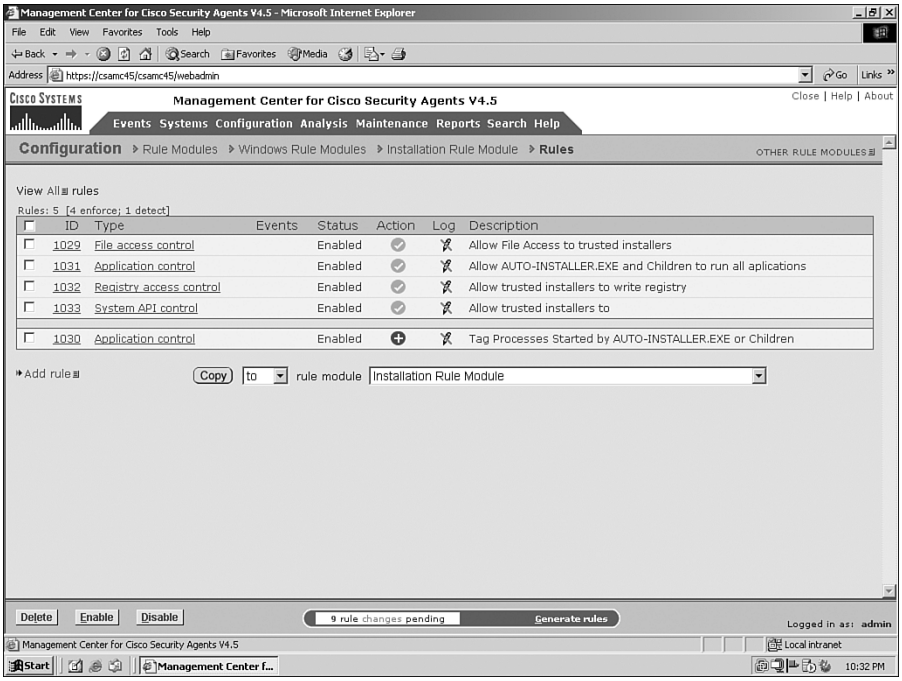
(d) Add a Registry Access Control rule to allow the same two application classes to write to all registry locations.

(e) Add a System API Control rule that allows whatever temporary security controls you want to provide to the installers. Common installer System API rule violations could include: trapping keystrokes, accessing memory of other applications, injecting code into other applications, and accessing functions in data or stack.

Remember, this policy does not weaken the entire system, but just provides these specific installation applications the temporary rights they need to complete an installation without causing the CSA administrator the headache of providing updated policy every time a new installation occurs. The key to everything in the previous example is the dynamic application class and the amount of intelligence and control it provides you. Mastering this concept ensures a successful deployment. The list of rules we added, as seen in Figure

9-17, might not be a complete configuration for every installation application on the market, but should provide you a start in creating your own policy.

Figure 9-17 *List of Rules from the Installation Policy*



Forensics

You can use the CSA and various rules and features of the product to report behavior you want to monitor on certain systems. The two methods used are: Monitor Rules and Application Behavior Investigation. The remaining portion of the chapter discusses these two methods.

Monitor Rules

You can create rules that do not enforce any security Allow or Deny actions but rather log an event only when the matching rule is triggered. These rules use an action of Monitor. You can create any type of rule with this type of action. The following are examples of rules that might be useful:

- Monitor execution of a specific application, such as a known P2P, Instant Messenger, or other unapproved application.

- Monitor FTP, TFTP, IRC, and other connections that should not leave your corporate network.
- Monitor file access of certain directories and file types.

You can use these rules when needed or create a Rule Module that includes several different types of rules with the Monitor action, each tied to a specific empty application class. Using this approach, you can add an executable to this application class when you locate a process you want to monitor, and you instantly begin to receive forensic data about the process after the next rule generation. This can provide you a Honey-Pot approach to monitoring that is available to you anywhere in the deployment at any time.

Application Behavior Investigation

The CSA product also provides a mechanism for monitoring a process natively named Application Behavior Investigation. This is configured by selecting **Analysis>Application Behavior Investigation>Windows Behavior Analyses**. Select **New** to create an investigation. You define the matching application class and the host the investigation should target. After completion and after a specified period or number of executions, you receive a report that displays all the network interaction, file interaction, COM object interaction, and registry interaction of that process. This can be a useful way to collect data about what a process does as part of research and also prior to creating and application control policy for this software.

Summary

Creating your own policies is a major part of operating a successful CSA deployment. To accomplish this, you must thoroughly understand the components available to you and the methods of research available. Understanding the rule types and the events caused by those rules helps you move forward in your deployment and perform day-to-day support. A solid grasp of the fundamentals and advanced components not only makes you an effective administrator but also an efficient one.