Writing Custom Signatures for the Cisco Intrusion Prevention System

Contents

Introduction

This paper provides instruction in creating signatures for the Cisco Intrusion Prevention System (IPS).

Signature Development Lifecycle

The signature development lifecycle has the following steps:

- Research the vulnerability
- Develop the signature
- Test the signature

Research the Vulnerability

During the vulnerability research step, the signature developer essentially thinks like a potential attacker and gains the knowledge necessary to exploit the vulnerability. The idea is that understanding the vulnerability will allow the signature developer to protect against attempts to exploit that vulnerability. Ideally, the signature developer will have actual exploits and have knowledge of the full range of attack vectors. Various sources can be used to gather this information. These include the following:

- Vendor security advisories
- Security forums (such as Bugtraq and Full Disclosure)
- Reverse engineering vendor patches
- Tools such as Metasploit

Determine Patterns in Malicious Traffic
After the developer has an understanding of an attack, the information is used to get insight into how malicious traffic looks as it traverses the network. At the end of this step, the developer should have answers to these types of questions:

- What protocols are relevant to the attack?
- What fields of the relevant protocols are used in attacks?
- What values are used for these fields when an attack is being launched?

False positives: The developer should determine whether the malicious traffic appears in normal network communication. If it does, there is the potential for the signature to generate an alert that is a *false positive*. A false positive exists if a signature that is written to detect a particular threat generates alerts based on benign traffic.

False negatives: Ideally, a signature will detect all possible attacks that successfully exploit the relevant threat. If the signature does not detect a particular attack, missing the attack is termed a *false negative* for that threat.

Develop the Signature

IPS Engine Overview
Cisco IPS uses signature engines to inspect traffic. A signature engine decodes protocols and exposes elements of those protocols to signature developers through engine parameters. Signature developers can use engine parameters to inspect specific protocol fields to detect malicious traffic. Perhaps the most important signature engine parameters are those that use regular expressions (regexes). This document includes a section that discusses Cisco IPS regular expressions. The guide *Installing and Using Cisco Intrusion Prevention System Device Manager 7.0* provides in-depth documentation of Signature Engines.

The three most-used engines are the String TCP, Atomic IP, and Service HTTP engines. The following sections provide details about these engines and summarize the most important information you will need to get started producing signatures quickly.

String TCP Engine

The String TCP engine allows users to inspect TCP payload. The following table documents other useful engine parameters:

Table 1. Select String TCP Engine Parameters

Parameter	Description
Direction	This parameter can be set to *to-service* or *from-service*. If the goal is to inspect traffic that is going to a host that sends the initial SYN when establishing a TCP connection, **Direction** should be set to *from-service*. If the goal is to inspect traffic that is coming from the host that receives the initial SYN, the direction should be set to *to-service*.
Min Match Length	This parameter is used to ensure that the TCP payload that matches a given regex is greater than a given length. **Min Match Length** measures the length, in bytes, from where a regex starts matching the data to where the match ends.
Exact Match Offset	This parameter is used to specify the exact offset into a TCP stream where a match ends. This is a zero-based offset. In fact, all the offsets are zero based.
Min Match Offset	This parameter is used to specify an offset into the stream that is being inspected. Signatures will generate an alert if the portion of the stream matched by the regex is located after this offset.
Max Match Offset	This parameter is used to specify an offset into the stream being inspected. Signatures will generate an alert if the portion of the steam matched by the regex is located before this offset.

Atomic IP Engine

The Atomic IP engine allows users to inspect the headers and payload of an IP packet. The following table documents useful engine parameters:

Table 2. Select Atomic IP Engine Parameters

Parameter	Description
Specify Layer 4 Protocol	This parameter specifies which Layer 4 protocol to inspect. Options include *TCP*, *UDP*, *ICMP*, and *Other*.

Specify Payload Inspection	If this parameter is set to *yes*, users can specify a regex, min match length, min match offset, and max match offset. These parameters function as documented in Table 1.
TCP Flags	See the following section
TCP Mask	See the following section

Atomic IP Engine Parameters: Specify TCP Flag and TCP Mask

You can specify TCP flags and masks in Cisco IPS versions 4.x and 5.x. The formula is as follows:

(packet & mask) == flags

The TCP flags from the packet in question are logically ANDed with the **Mask** parameter and compared against the **TcpFlags** parameter.

In brief, this formula means

- If a flag is not set in the mask, there is no examination for the presence of the flag in the packet.
- If a flag is set in the mask and not in the flags, the flag must not be present in the packet.
- If a flag is set in both the mask and flags, the flag must be present in the packet.

Example: Match All packets with SYN and ACK Flags Set

TcpFlags == SYN|ACK

Mask == SYN|ACK

SYN and ACK are set in the mask and flags because they are required to be set. The signature would fire if any other flags were set in the packet.

Example: Explicitly Match Packets with Just the SYN and ACK Flags Set

TcpFlags == SYN|ACK

Mask == SYN|ACK|RST|URG|PSH|FIN

This mask is necessary to force only the SYN and ACK to be set.

Service HTTP Engine

The Service HTTP engine is used to inspect HTTP headers. It currently does not inspect the content transported over HTTP. The following table documents useful engine parameters:

Table 3. Select Service HTTP Engine Parameters

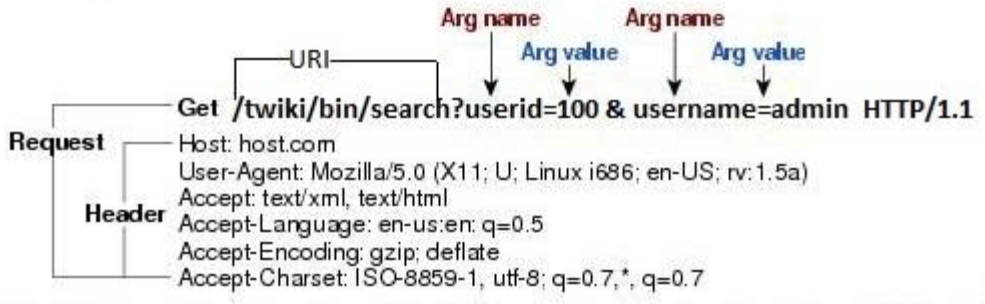
Parameter	Description
Specify Max URI Field Length, Specify Max Arg Field Length, Specify Max Header Field Length, Specify Max Request Field Length	These parameters are used to apply a length limit to the URI field, Arg field, entire HTTP header, and Request fields. See Figure 1 for a visual representation.
Specify URI Regex, Specify Arg Name Regex, Specify Header Regex, Specify Request Regex	These parameters are used to apply a regular expression to the URI, Arg Name, Header, and Request portion of an HTTP header. See Figure 1 for a visual representation.

Figure 1 shows how the Service HTTP engine parameters are used. Individual arguments are separated by an ampersand (&), whereas the argument name and value are separated by one equal sign (=).

Figure 1. HTTP Engine Parameters

User Input: http://10.20.35.6/twiki/bin/search?userid=100&username=admin

Browser output:



Other Engines

Cisco IPS provides numerous additional engines. Examples include Service DNS, Service SMB Advanced, and Fixed TCP. For a full list of engines, see the Signature Engines section of *Installing and Using Cisco Intrusion Prevention System Device Manager 7.0*.

Choosing the Best Engine

At times, it is obvious which engine should be used. For example, if a signature is being written to detect an exploit that targets a vulnerability in the SMB protocol, it make sense to use the Service SMB Advanced engine. However, sometimes it is possible to use several engines to detect certain threats. Following are some questions that will aid this decision-making process:

Q. Can an engine inspect the type of traffic necessary?
A. If the engine cannot detect the type of traffic in question, it is eliminated from the list of possible engines. For example, if a vulnerability is exploited via UDP, the String TCP engine is irrelevant because it cannot inspect UDP traffic.

Q. Does an engine provide access to a particular protocol's fields via engine parameters?
A. If the engine decodes a particular protocol, it may make it easier for the signature developer to write a particular signature. For example, a signature can be written to detect a pattern in SMB traffic using the String TCP traffic because TCP is often used to transport SMB. However, the SMB Advanced traffic exposes particular portions of the SMB protocol via engine parameters. Thus, it may be easier to use the SMB Advanced engine instead of the String TCP engine.

Regular Expressions
Cisco IPS uses regular expressions to define rules to match malicious or unwanted behavior over the network. All the different engines in Cisco IPS support regular expressions. Here are some basic examples.

Square Brackets

To match the exact word *hello*, you would use the following string:

```
hello
```

Which expression would you use, however, if the first character of this word could be uppercase or lowercase?

```
[hH]ello
```

Square brackets allow you to specify different possibilities within a single expression. In this case, *[hH]ello* basically means *hello* or *Hello*. These brackets will match if one single character out of the set specified in the brackets matches the expression.

To match the word *hello* in any form, no matter whether any single letter uses uppercase or lowercase, the regular expression has to be modified. The correct form would be as follows:

```
[hH][eE][lL][lL][oO]
```

This expression will match *heLLo* or *HeLlO* or *hELLO* (and so on).

Could you just use the expression *[hHeElLlLoO]*? Because square brackets and their content can represent only a single character, the content of these brackets would match the single letter *h*, *e*, *l*, or *o* spelled using uppercase or lowercase. Therefore, this expression is not the same as *[hH][eE][lL][lL][oO]*.

The OR Operator

You can also match either of two variants, such as *document* or *file*, with a single expression. This is where the OR operator comes into play. This operator is represented by the single character |, also called a *pipe*. It allows you to match either one possibility or the other. The regular expression in question would look like this: *document|file*.

The NOT Operator

Sometimes you need to specifically exclude one or more characters in a regular expression. Common examples are signatures that may trigger only if a certain character is not found at a certain position or within a certain range. To exclude one or more characters, you can use the circumflex character. As a first example, here is a regular expression for the letter *A* not followed by the letter *B*: *A[^B]*.
You can also use this expression to deny more than one letter. If *A* must not be followed by *C* or *D*, here is the expression: *A[^CD]*.

Ranges

How do you write a regular expression for a single letter that is followed by a number that is 0 through 9?

First Try

Because you do not know which letter of the alphabet will be followed by which number, you will have to match all of them. For the sake of convenience, the example will be lowercase. Here is one possibility to accomplish this goal:

```
[abcdefghijklmnopqrstuvwxyz][0123456789]
```

This expression will do the job. Two sets of brackets equal two characters, one a lowercase letter, one a number that is 0 through 9. The expression is correct, just not very convenient.

Second Try

Regular expressions allow for the definition of ranges of characters or numbers, which comes in very handy for matching an expression like this one. Instead of *[0123456789]*, the following expression will do the same job: *[0-9]*.
This technique will also work for letters. Here is the alphabet, in short: *[a-z]*.
And here is a regular expression that will match the same two-character string as the one in the first try: *[a-z][0-9]*.

How would you match the letter if it could be uppercase or lowercase? You need to use a single square bracket set to match the lowercase and uppercase alphabet. Here is a modification of the first part of the expression that matches the new requirements: *[a-zA-Z]*.

Grouping Expressions

A preceding example explained the OR operator and how to apply it to match one word or another. But how would you distinguish between parts of words? How would you make sure the regex parser knows which fragments to use with the OR operator? Does *abc|def* mean *abc* OR *def*? Or does it mean *ab* followed by the letter *c* OR *d* followed by *ef*? What if the requirement was to match *a* when it is followed by *bc* OR *de*, followed in turn by *f*?

Parentheses can clarify expressions. They allow you to group expressions to clearly define where one expression ends and another starts. This way you can safely and easily differentiate between the different sets:

- *(abc)|(def)* equals *abc* OR *def*
- *ab(c|d)ef* equals *ab* followed by the letter *c* OR *d* followed by *ef*
- *a((bc)|(de))f* equals *a* followed by *bc* OR *de* followed in turn by *f*

Escaping Characters

So far it was shown that parentheses, square brackets, and the pipe are special characters. But what if you need to match a string that contains one of these characters? There are three possible solutions:

Include the character to match in square brackets: Examples follow:

- *[|]* matches the pipe. The expression *abc[|]def* matches the string *abc|def*.
- *[[]* matches the opening bracket. The expression *AA[[]BB* matches the string *AA[BB*.

Escape the character as you would in UNIX by using the backslash character: Examples follow:

- \| matches the pipe. The expression *abc*\|*def* matches the string *abc|def*
- \[matches the opening bracket. The expression AA\[BB matches the string *AA[BB*.

Use the hexadecimal representation: Every character has a hexadecimal number, commonly referred to as the hex code, that may be used at any time instead of the real character. To notify the regex parser of upcoming hexadecimal representations, place the two-letter combination *x* before each

hex code. For example, *\x41\x42\x43* is the same as *ABC*.

Counting

To define several instances of the same expression, you can use the expression several times in a row. To match the letter *A* five times, this will do: *AAAAA*. As will this: *\x41\x41\x41\x41\x41*. Or this: (*AAAAA*). But there is an easier way to tell the regex parser to look for an expression more than once: braces. The expression can easily be replaced by *A{5}* or by *[A]{5}*.

Continuation

What if the length of a certain string was unknown? In this example, you know only that the string starts with an *A*, it is followed by one or more *A*s, and it ends with the letter *B*. You may try this:

```
AAB  | AAAB | AAAAB | AAAAAB | ...
```

Not very effective, right? That is why there are two special wildcard operators: the plus (+) and the asterisk (*). The plus operator signifies that one or more of the previous expressions will follow, whereas the asterisk means that zero or more of those expressions will follow. Keep in mind that these operators will not stop matching, so be sure to terminate them by adding a terminal expression (such as the *B* in the following examples). In this case, the following expressions match all the preceding examples in this section, and many more:

- *A+B* matches at least one *A*, or many more, followed by *B*
- *AAA*B* matches *AA* followed by zero or more *A*s followed by *B*

Whereas the expression *a** means zero or more, up to many, occurrences of *a*, the expression *a+* means that there is at least one *a*, but there could also be more. The expression *a+* is equivalent to *aa**.

Regular expressions that end with a plus or asterisk should be set to **fire-once** to ensure that they do not go off for every single matching following character after they have fired the first time.

Potentially Missing Expressions

Sometimes a regular expression could be written to match data sets that differ by a slight margin, if only one part of the expression could be rendered ineffective in certain cases. This problem can be solved by applying the question mark (?) operator. If used in an expression, this operator indicates that the expression right before the question mark might be applied, but it does not have to be.

Here is a simple example. Anton has a sister, Antonia. The only way of writing one expression that properly matches both their names is with the question mark operator because it allows the matching of the last two letters in Antonia's name to be conditional. Here is an example expression:

```
Anton(ia)?
```

This expression matches *Antonia* as well as *Anton* because the characters in the parentheses do not have to be applied for a match to occur.

Signature Configuration on Cisco IPS

IPS Device Manager Overview

There are multiple options for modifying the signature configuration on the IPS sensor. The easiest method to begin with uses the IPS Device Manager (IDM) utility.

IDM is a Java-based web tool that implements a graphical user interface (GUI) that you can use to configure a Cisco IPS device.

The first step to using IDM is installing the software. Begin the installation by directing a Java-enabled web browser to the Cisco IPS device on port 443:

```
https://<IP of IPS device>/
```

You will see a web page that offers to install IDM.

Figure 2. IDM Installation Web Page

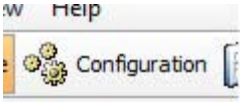
Click **Run IDM**. You will see a prompt to perform a Java Web Start installation of the IDM application. For the download to work, ensure that Java Runtime Environment (JRE) is installed. One place to acquire this software is the Sun area of the Oracle website. (http://www.oracle.com/us/sun/).

After Java Web Start runs, IDM will be installed on your computer and a Launcher login screen will appear (see Figure 3). The credentials for this prompt were configured when the IPS device itself was set up.

Figure 3. IDM Launcher Login Screen

After you complete the login process, you will see the IDM interface. Initially the dashboard will be displayed. This dashboard shows statistics about the health and configuration of the IPS device. Click **Configuration** in the top menu bar to move to the IPS device configuration screen.

Figure 4. Configuration Button in the IDM Dashboard

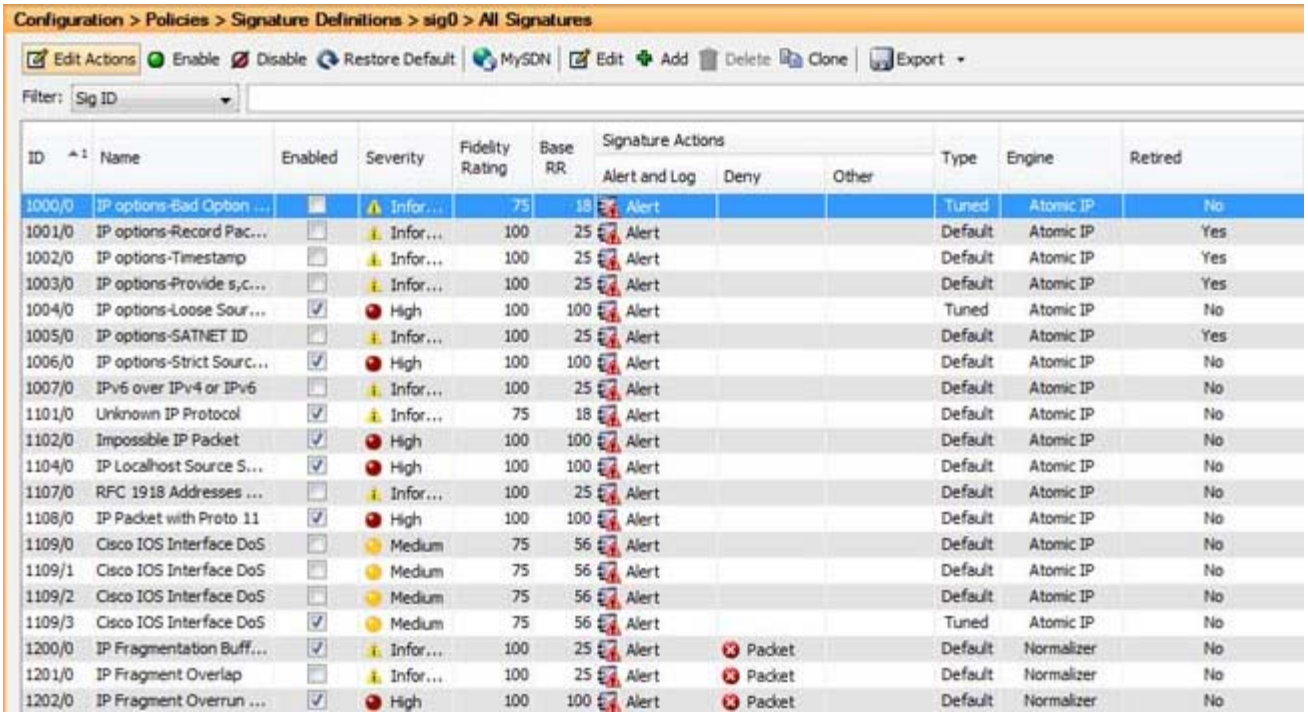


From here, click **Policies** in the bottom left (as shown in Figure 5), then click **All Signatures** to navigate to the signature page.

Figure 5. Policies Button in the IPS Device Configuration Screen

The signature view (see Figure 6) allows you to perform a variety of tasks on the existing signature set, including enabling/disabling signatures and adding/deleting signatures. You can also use the **Clone** button to create a copy of an existing signature.

Figure 6. IDM Signature Page



ID	Name	Enabled	Severity	Fidelity Rating	Base RR	Signature Actions			Type	Engine	Retired
						Alert and Log	Deny	Other			
1000/0	IP options-Bad Option ...	☐	Infor...	75	18	Alert			Tuned	Atomic IP	No
1001/0	IP options-Record Pac...	☐	Infor...	100	25	Alert			Default	Atomic IP	Yes
1002/0	IP options-Timestamp	☐	Infor...	100	25	Alert			Default	Atomic IP	Yes
1003/0	IP options-Provide s,c...	☐	Infor...	100	25	Alert			Default	Atomic IP	Yes
1004/0	IP options-Loose Sour...	☑	High	100	100	Alert			Tuned	Atomic IP	No
1005/0	IP options-SATNET ID	☐	Infor...	100	25	Alert			Default	Atomic IP	Yes
1006/0	IP options-Strict Sourc...	☑	High	100	100	Alert			Default	Atomic IP	No
1007/0	IPv6 over IPv4 or IPv6	☐	Infor...	100	25	Alert			Default	Atomic IP	No
1101/0	Unknown IP Protocol	☑	Infor...	75	18	Alert			Default	Atomic IP	No
1102/0	Impossible IP Packet	☑	High	100	100	Alert			Default	Atomic IP	No
1104/0	IP Localhost Source S...	☑	High	100	100	Alert			Default	Atomic IP	No
1107/0	RFC 1918 Addresses ...	☐	Infor...	100	25	Alert			Default	Atomic IP	No
1108/0	IP Packet with Proto 11	☑	High	100	100	Alert			Default	Atomic IP	No
1109/0	Cisco IOS Interface DoS	☐	Medium	75	56	Alert			Default	Atomic IP	No
1109/1	Cisco IOS Interface DoS	☐	Medium	75	56	Alert			Default	Atomic IP	No
1109/2	Cisco IOS Interface DoS	☐	Medium	75	56	Alert			Default	Atomic IP	No
1109/3	Cisco IOS Interface DoS	☑	Medium	75	56	Alert			Tuned	Atomic IP	No
1200/0	IP Fragmentation Buff...	☑	Infor...	100	25	Alert	Packet		Default	Normalizer	No
1201/0	IP Fragment Overlap	☐	Infor...	100	25	Alert	Packet		Default	Normalizer	No
1202/0	IP Fragment Overrun ...	☑	High	100	100	Alert	Packet		Default	Normalizer	No

(Click for larger image)

Signature Creation Demonstration

Now that you are familiar with the basics of IDM, you can look at the process of using it to enter a signature onto the IPS sensor itself.

For the sake of this demonstration, you will enter a signature that looks for the regular expression *[A-D][A-B][A-C]a+* sent to a server on TCP port 6060. You will also restrict this signature to looking at the first 5,000 bytes of the TCP stream.

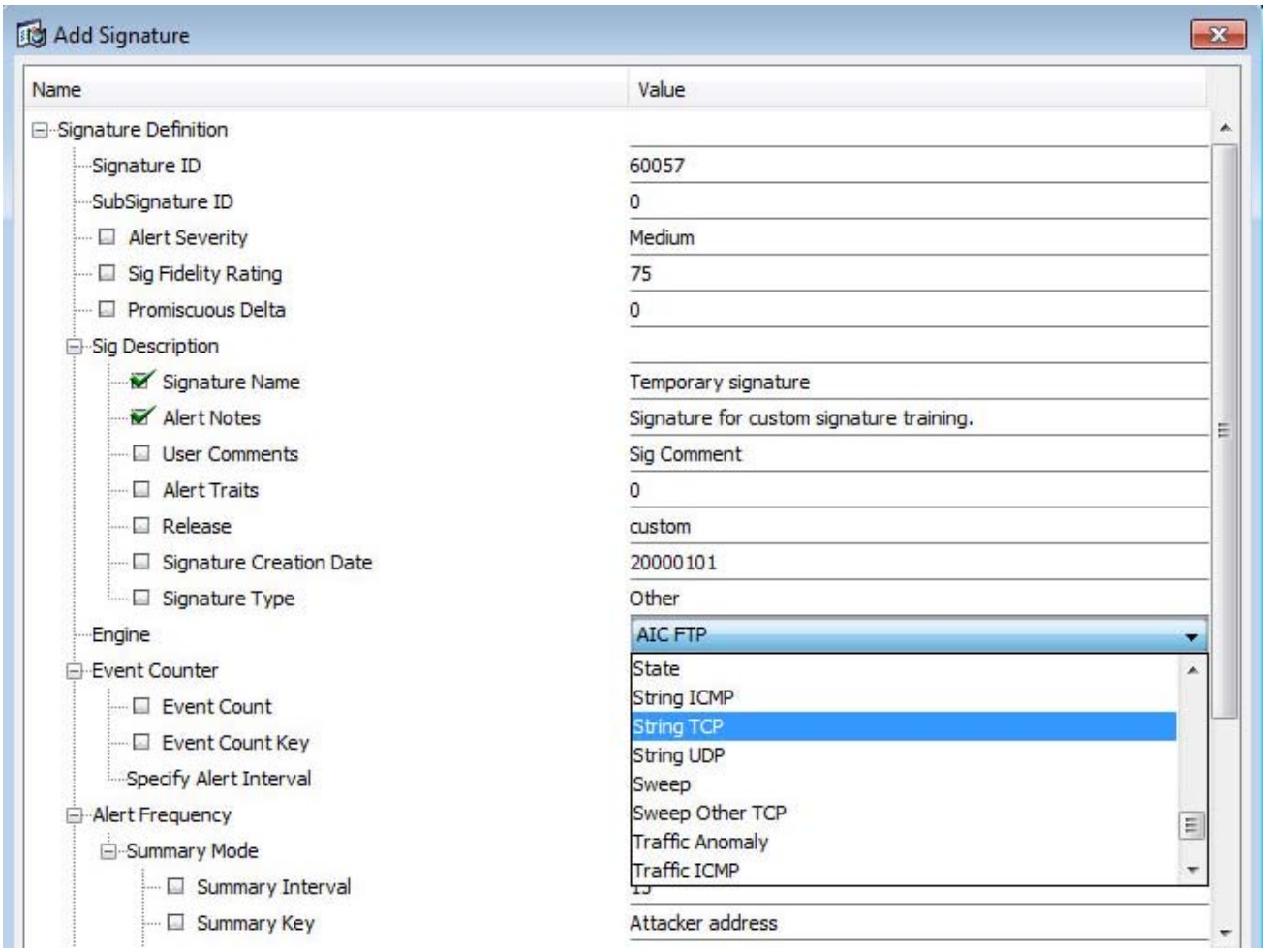
To begin entering this signature, click **Add** on the signature page.

Figure 7. Add Button in IDM Signature Page

You will see a dialog box that contains fields for each parameter of the signature. Begin by populating the **Signature Name** and **Alert Notes** fields with the data that should be displayed when the signature fires.

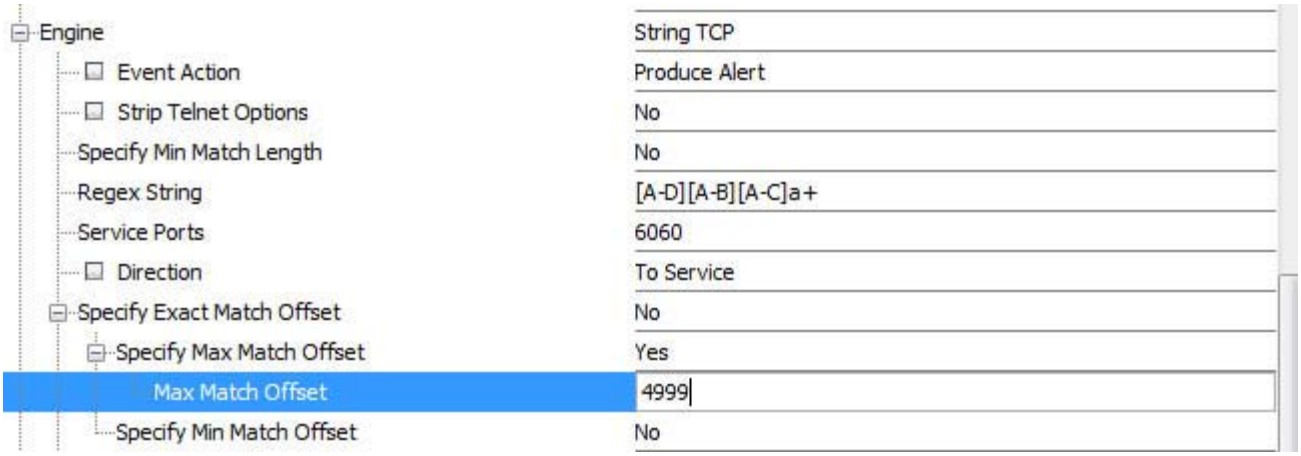
The next step is to select the required engine. You will then see the appropriate fields for the selected engine. For the signature described for this demonstration, choose the String TCP engine (see Figure 8).

Figure 8. Choosing the String TCP Engine in the Add Signature Dialog Box



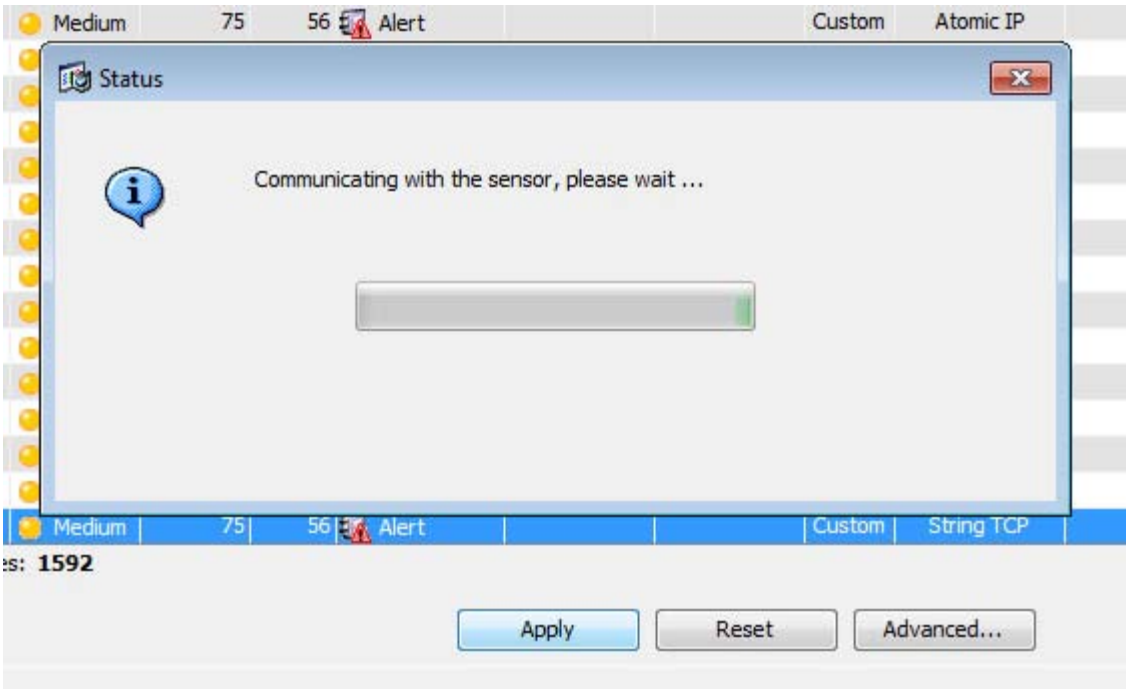
Now that the fields for the engine are visible, you can populate them with the values for the demonstration signature. Enter the regular expression into the **Regex String** field, and enter the port number in **Service Ports**. Leave **Direction** set to the default (To Service) because the demonstration signature should fire on traffic that is traveling to TCP port 6060. To match the first 5,000 bytes, set **Max Match Offset** to the value 4999, accounting for the fact that this field offsets from zero. Figure 9 shows the completed fields.

Figure 9. Engine Settings in the Add Signature Dialog Box



After you have entered the engine parameters, click **OK**. The dialog box will close, but the signature will not be pushed to the actual sensor until you click **Apply**. After you apply the changes, you will see the following message window, indicating that the signature is being pushed to the sensor.

Figure 10. Applying the Signature to Push It to the Sensor



After the update process has finished, the signature has been applied to the sensor and is now active.

IPS Command-Line Interface

For some cases, it may be faster to use the IPS command-line interface (CLI) instead of launching IDM to create or modify a signature. The CLI also allows you to perform batch processing of a large number of custom signatures, which is not covered in this paper. If you have some knowledge of the Cisco IOS CLI, these processes should not be difficult.

Entering an Individual Signature Using the CLI

Access the IPS CLI by entering the IP address of the IPS device in your SSH client (such as OpenSSH or Putty). Log in as the user *cisco*.

After you have logged in to the device, enter the following command to put the CLI into configuration mode:

```
ipsdevice# conf t
ipsdevice(config)#
```

Now you can begin to enter the signature. This demonstration will use the same signature described in the IDM example. The signature uses the String TCP engine and looks for the regular expression *[A-D][A-B][A-C]a*+ traveling to TCP port 6060. The Max Match Offset will be 4999 again.

To begin the process of entering the signature, enter the context of the appropriate signature definition. By default, this context is *sig0*, which is entered with the following command:

```
ipsdevice(config)# service signature-definition sig0
ipsdevice(config-sig)#
```

Next, enter a signature ID for the signature. For this example, use 61101, which is unlikely to be in use. Use the following command to begin editing signature 61101:

```
ipsdevice(config-sig)# signatures 61101 0
```

Now that you are in the context of signature 61101, you need to populate the attributes for this signature, starting with the signature name. The name is in the sig-description context, so enter the **sig-description** command, then enter the **sig-name** command followed by the signature name.

```
ipsdevice(config-sig-sig)# sig-description
ipsdevice(config-sig-sig-sig)# sig-name Test Signature
ipsdevice(config-sig-sig-sig)# exit
```

Next, select the engine for the signature and populate its parameters. In this case, you will use the String TCP engine:

```
ipsdevice(config-sig-sig)# engine string-tcp
```

You can then enter the signature parameters:

```
ipsdevice(config-sig-sig-str)# regex-string [A-D][A-B][A-C]a+
ipsdevice(config-sig-sig-str)# service-ports 6060
ipsdevice(config-sig-sig-str)# specify-exact-match-offset no
ipsdevice(config-sig-sig-str-no)# specify-max-match-offset yes
ipsdevice(config-sig-sig-str-no-yes)# max-match-offset 4999
```

Finally, enter the **exit** command multiple times to exit each step in the process:

```
ipsdevice(config-sig-sig-str-no-yes)# exit
ipsdevice(config-sig-sig-str-no)# exit
ipsdevice(config-sig-sig-str)# exit
ipsdevice(config-sig-sig)# exit
ipsdevice(config-sig)# exit
ipsdevice(config)# exit
```

The IPS device will prompt you to save the changes. If you reply *yes* to this prompt, the signature will be updated on the device.

```
Apply Changes?[yes]: yes
Processing config: /
```

Cloning a Signature

Administrators often find the need to modify a signature to meet the needs of a specific network, such as to reduce false positives or false negatives. In such cases, the first approach should be to fine tune signature parameters such as event action filters and override policies. If these tunings are not sufficient, the last action that is available is to modify a signature. By default, signature parameters such as the regular expression cannot be modified. The signature must first be cloned in order to modify such signature parameters. The original signature can be retired or disabled if it is determined that it is no longer required.

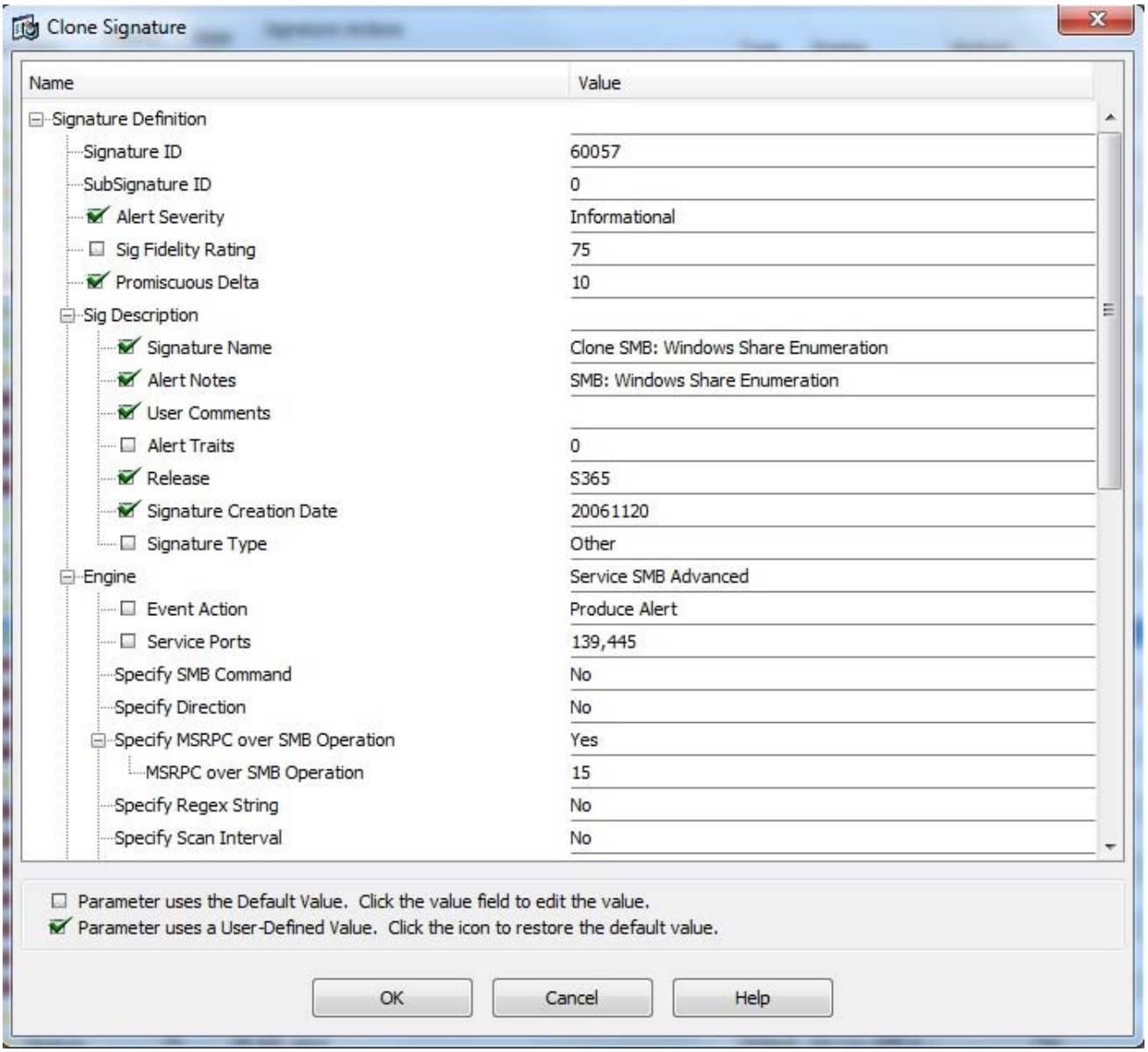
How to Clone a Signature

Cloning a signature results in an exact copy of the signature. To clone a signature, go to **Configuration > Policies > Signature Definitions**, select the signature that needs to be cloned, and click the **Clone** button (as shown in Figure 11).

Figure 11. Clone a Signature

You will see a new window with the signature details of the signature that is being cloned (as shown in Figure 12).

Figure 12. Clone Signature Parameters



"Clone" will be appended to the new signature name by default. The signature ID field defaults to the custom signature ID range (greater than 60000). This value can be changed to any unique value within this range. Make changes to the sub sig id if needed. The rest of the signature parameters—including Signature Engine, Regex, Offsets, and all other signature parameters—will be a copy of the signature being cloned. Make changes as necessary and click **OK** to close the dialog box. To push the changes to the sensor, click **Apply**. This will push the new configuration to the sensor.

The following fields cannot be cloned:

- Obsoletes
- Protected parameters
- Hidden parameters
- Cisco Security Monitoring, Analysis, and Response System (MARS)category

If a signature that is retired and disabled by default is cloned, the new signature will be enabled and unretired.

Hidden and Protected Signature Parameters

Some of the default signature parameters are hidden and protected. These parameters are not visible through IDM or the IPS CLI. Such signature parameters cannot be modified or cloned. This limitation applies to components of a meta signature as well as components of the Multi String engine.

Test the Signature

Testing validates the signature. Many of the testing steps are performed iteratively with signature development steps. Cisco strongly recommends using **produce-verbose-alert** *event-action* for custom signatures, especially for the testing phase.

For illustrative purposes, this section examines a signature that detects traffic that attempts to contact command-and-control servers for a botnet.

Hosts that are infected with the associated malware create an HTTP header that looks like the following:

```
GET /image.png HTTP/1.0
User-Agent: evil-software/3.2
Accept: */*
Host: example.org
Connection: Keep-Alive
```

Researching this issue revealed that hosts that are infected with the malware send an **HTTP GET** request to example.org with a User-Agent string of *evil-software/3.2*. Thus, the (example) signature for detecting this traffic would use the Service HTTP engine and have the request regex set to *User-Agent:\x20evil-software\x2f3\x2e2*. The following example is the complete signature configuration displayed from the IPS CLI. You could also view this information in IDM.

```
IPS-4240# configure terminal
IPS-4240(config)# service signature-definition sig0

IPS-4240(config-sig)# signatures 60023 0
IPS-4240(config-sig-sig)# show settings
   sig-id: 60023
   subsig-id: 0
   -----------------------------------------------
      alert-severity: high default: medium
      sig-fidelity-rating: 95 default: 75
      promisc-delta: 0 <defaulted>
      sig-description
      -----------------------------------------------
         sig-name: Botnet command and control traffic default: My Sig
         sig-string-info: User-Agent: evil-software/3.2 default: My Sig Info
         sig-comment: Sig Comment <defaulted>
         alert-traits: 0 <defaulted>
         release: custom <defaulted>
         sig-creation-date: 20000101 <defaulted>
         sig-type: Other <defaulted>
      -----------------------------------------------
      engine
      -----------------------------------------------
         service-http
         -----------------------------------------------
            event-action: produce-alert <defaulted>
            de-obfuscate: true <defaulted>
            max-field-sizes
            -----------------------------------------------
               specify-max-uri-field-length
               -----------------------------------------------
                  no
                  -----------------------------------------------
                  -----------------------------------------------
               -----------------------------------------------
               specify-max-arg-field-length
               -----------------------------------------------
                  no
                  -----------------------------------------------
                  -----------------------------------------------
               -----------------------------------------------
               specify-max-header-field-length
               -----------------------------------------------
                  no
                  -----------------------------------------------
                  -----------------------------------------------
               -----------------------------------------------
               specify-max-request-length
               -----------------------------------------------
                  no
                  -----------------------------------------------
                  -----------------------------------------------
               -----------------------------------------------
            -----------------------------------------------
            regex
            -----------------------------------------------
               specify-uri-regex
               -----------------------------------------------
                  no
                  -----------------------------------------------
                  -----------------------------------------------
               -----------------------------------------------
               specify-arg-name-regex
               -----------------------------------------------
                  no
                  -----------------------------------------------
```

```
                  ------------------------------------------------
               ------------------------------------------------
               specify-header-regex
               ------------------------------------------------
                  no
                  ------------------------------------------------
                  ------------------------------------------------
               ------------------------------------------------
               specify-request-regex
               ------------------------------------------------
                  yes
                  ------------------------------------------------
                     request-regex: User-Agent:\x20evil-software\x2f3\x2e2
                     specify-min-request-match-length
                     ------------------------------------------------
                        no
                        ------------------------------------------------
                        ------------------------------------------------
                     ------------------------------------------------
                  ------------------------------------------------
               ------------------------------------------------
            ------------------------------------------------
            service-ports: #WEBPORTS
            swap-attacker-victim: true default: false
         ------------------------------------------------
      ------------------------------------------------
      event-counter
      ------------------------------------------------
         event-count: 1 <defaulted>
         event-count-key: AxBx default: Axxx
         specify-alert-interval
         ------------------------------------------------
            no
            ------------------------------------------------
            ------------------------------------------------
         ------------------------------------------------
      ------------------------------------------------
      alert-frequency
      ------------------------------------------------
         summary-mode
         ------------------------------------------------
            summarize
            ------------------------------------------------
               summary-interval: 15 <defaulted>
               summary-key: Axxx <defaulted>
               specify-global-summary-threshold
               ------------------------------------------------
                  no
                  ------------------------------------------------
                  ------------------------------------------------
               ------------------------------------------------
            ------------------------------------------------
         ------------------------------------------------
      ------------------------------------------------
      status
      ------------------------------------------------
         enabled: true <defaulted>
         retired: false <defaulted>
         obsoletes (min: 0, max: 65535, current: 0)
         ------------------------------------------------
         ------------------------------------------------
      ------------------------------------------------
      vulnerable-os: general-os <defaulted>
      specify-mars-category
      ------------------------------------------------
         yes
         ------------------------------------------------
            mars-category: Info/Misc <defaulted>
         ------------------------------------------------
      ------------------------------------------------
   ------------------------------------------------
IPS-4240(config-sig-sig)#
```

Unit Testing
Unit testing entails creating the signature on a sensor and replaying traffic through the sensor to ensure that the signature fires as expected.

Cisco captured traffic from a host infected with the malware in the lab. It is also possible to simulate additional pcap files for testing purposes. For example, you can use **wget** *-U "evil-software/3.2" example.org/image.png* to send an HTTP request to example.org with the User-Agent string set to

evil-software/3.2. It is often necessary to create traffic samples to test signatures. The following code snippet shows how to generate and capture this traffic.

```
$ tshark -w sample_traffic.pcap host example.org
```

Run **wget** from a different terminal:

```
$ wget -U "evil-software/3.2" example.org/image.png
```

When the **wget** command execution is complete, TShark will have created a pcap file that contains sample traffic you can use to test the signature. You can use the **tcpreplay** command to replay the test traffic to the IPS device to test the signature:

```
$ tcpreplay -i eth0 sample_traffic.pcap
```

You can check the IPS device to verify that the signature fired:

```
IPS-4240# show events alert

evIdsAlert: eventId=1287597699201549604 severity=high vendor=Cisco
  originator:
    hostId: IPS-4240
    appName: sensorApp
    appInstanceId: 9803
  time: 2010/11/12 22:42:18 2010/11/12 22:42:18 UTC
  signature: description=Botnet command and control traffic id=60023
       created=20000101 type=other version=custom
    subsigId: 0
    sigDetails: User-Agent: evil-software/3.2
    marsCategory: Info/Misc
  interfaceGroup: vs0
  vlan: 0
  participants:
    attacker:
      addr: locality=OUT 192.0.32.10
      port: 80
    target:
      addr: locality=OUT 64.101.177.162
      port: 51099
      os: idSource=unknown relevance=relevant type=unknown
  actions:
    denyPacketRequestedNotPerformed: true
    denyFlowRequestedNotPerformed: true
  context:
    fromTarget:
000000  47 45 54 20 2F 69 6D 61  67 65 2E 70 6E 67 20 48  GET /image.png H
000010  54 54 50 2F 31 2E 30 0D  0A 55 73 65 72 2D 41 67  TTP/1.0..User-Ag
000020  65 6E 74 3A 20 65 76 69  6C 2D 73 6F 66 74 77 61  ent: evil-softwa
000030  72 65 2F 33 2E 32 0D 0A  41 63 63 65 70 74 3A 20  re/3.2..Accept:
000040  2A 2F 2A 0D 0A 48 6F 73  74 3A 20 65 78 61 6D 70  */*..Host: examp
000050  6C 65 2E 6F 72 67 0D 0A  43 6F 6E 6E 65 63 74 69  le.org..Connecti
000060  6F 6E 3A 20 4B 65 65 70  2D 41 6C 69 76 65 0D 0A  on: Keep-Alive..
000070  0D 0A                                            ..
  riskRatingValue: attackRelevanceRating=relevant targetValueRating=medium 100
  threatRatingValue: 100
  interface: ge0_0
  protocol: tcp
```

Unit testing can also involve testing for false positives. A false positive occurs when a signature fires an alert for traffic that is not malicious. To test for false positives, generate traffic samples for known benign traffic. This testing is particularly useful for signatures that detect malicious traffic that closely resembles benign traffic.

Real Network Traffic Test

Before declaring that a signature is ready for production use, it is wise to deploy it to IPS devices that inspect real-world traffic. This test gives signature developers an opportunity to tune signatures if needed and provides an additional layer of false-positive testing.

False Positive

A false positive occurs when legitimate network activity is interpreted and reported as an attack. This happens when network activity meets criteria that were specified to identify an attack. A false positive can be addressed by tuning the sensor configuration by disabling or retiring the signature or developing a higher-fidelity signature when possible.

False Positive Example

The original version of Signature 5916-0 detects the URL Handler vulnerability on return web ports with the following regex:

[\x22\x27](([Mm][Aa][Ii][Ll][Tt][Oo])|([Nn][Nn][Tt][Pp])|([Ss]?
[Nn][Ee][Ww][Ss])|([Tt][Ee][Ll][Nn][Ee][Tt])|([Ff][Ii][Rr][Ee][Ff][Oo][Xx][Uu][Rr][Ll])|([Ff][Ii][Rr][Ee][Ff][Oo][Xx][Hh][Tt][Mm][Ll]))[:][\x21\x23-\x26\x28-
\x7f]*[%]((\x30\x30)|(\x2e\x2e\x2f)

This is intended to detect attacks of the following form:

mailto:%00../../../../../../../../windows/system32/cmd.exe

The following alert is a false positive that triggers the preceding signature:

```
0000  43 6c 61 69 6d 20 66 6f  72 6d 73 20 6d 75 73 74   Claim fo rms must
0010  20 62 65 20 70 6f 73 74  6d 61 72 6b 65 64 20 62    be post marked b
0020  79 20 44 65 63 65 6d 62  65 72 20 33 30 2c 20 32   y Decemb er 30, 2
0030  30 31 30 20 61 6e 64 20  72 65 63 65 69 76 65 64   010 and  received
0040  20 62 79 0d 20 20 20 20  20 20 20 20 4a 61 6e 75    by.         Janu
0050  61 72 79 20 33 30 2c 20  32 30 31 31 2e 3c 62 72   ary 30,  2011.<br
0060  20 2f 3e 0d 20 20 20 20  20 20 20 20 37 2e 20 4f    />.         7. O
0070  72 64 65 72 73 20 61 72  65 20 6f 72 64 69 6e 61   rders ar e ordina
0080  72 69 6c 79 20 64 65 6c  69 76 65 72 65 64 20 77   rily del ivered w
0090  69 74 68 69 6e 20 36 2d  38 20 77 65 65 6b 73 2e   ithin 6- 8 weeks.
00a0  20 50 6c 65 61 73 65 20  63 61 6c 6c 20 31 2d 38    Please  call 1-8
00b0  37 37 2d 38 38 34 2d 30  30 34 38 0d 20 20 20 20   77-884-0 048.
00c0  20 20 20 20 6f 72 20 65  6d 61 69 6c 20 3c 61 20       or e mail <a
00d0  68 72 65 66 3d 22 6d 61  69 6c 74 6f 3a 25 32 30   href="ma ilto:%20
00e0  6d 69 74 73 75 62 69 73  68 69 40 6c 69 6e 6b 73   mitsubis hi@links
00f0  75 6e 6c 69 6d 69 74 65  64 2e 63 6f 6d 25 30 30   unlimite d.com%00
0100  fd 85 c1 c1 b1 a5 8d 85  d1 a5 bd b8 bd e1 b5 b0   ........ ........
0110  ed c4 f4 c0 b8 e4 b0 a8  bc a8 ed c4 f4 c0 b8 e0   ........ ........
0120  34 29 05 8d 8d 95 c1 d0  b5 31 85 b9 9d d5 85 9d   4)...... .1......
0130  94 e8 81 95 b8 b5 d5 cc  b1 95 b8 ed c4 f4 c0 b8   ........ ........
0140  d4 34 29 05 8d 8d 95 c1  d0 b5 15 b9 8d bd 91 a5   .4)..... ........
0150  b9 9c e8 81 9d e9 a5 c0  b1 91 95 99 b1 85 d1 94   ........ ........
```

The fix for the above false positive is to remove the percent sign (%) from the wildcarded character class, which results in a signature with higher fidelity.

The corrected regex follows:

[\x22\x27](([Mm][Aa][Ii][Ll][Tt][Oo])|([Nn][Nn][Tt][Pp])|([Ss]?
[Nn][Ee][Ww][Ss])|([Tt][Ee][Ll][Nn][Ee][Tt])|([Ff][Ii][Rr][Ee][Ff][Oo][Xx][Uu][Rr][Ll])|([Ff][Ii][Rr][Ee][Ff][Oo][Xx][Hh][Tt][Mm][Ll]))[:][\x21\x23\x24\x26\x28-
\x7f]*[%]((\x30\x30)|(\x2e\x2e\x2f))

Notice the change in the wild card character class from *[\x21\x23-\x26\x28-\x7f]* to *[\x21\x23\x24\x26\x28-\x7f]*.

False Negative

A false negative occurs when an attack is not detected. Tuning the sensor configurations will help to decrease the number of false negatives.

False Negative Example

Signature 11020-0 detects BitTorrent client activity. The regex of the initial signature follows:

^[\x13][Bb][Ii][Tt][Tt][Oo][Rr][Rr][Ee][Nn][Tt][\x20][Pp][Rr][Oo][Tt][Oo][Cc][Oo][Ll][\x00][\x00][\x00][\x00][\x00][\x00][\x00][\x00]

Because BitComet traffic does not contain *0*s after "bittorrent," the signature causes a false negative condition. The following is a modified version of the signature:

^[\x13][Bb][Ii][Tt][Tt][Oo][Rr][Rr][Ee][Nn][Tt][\x20][Pp][Rr][Oo][Tt][Oo][Cc][Oo][Ll]

Examples of Real Cisco IPS Signatures

The previous examples explained the techniques for developing a signature for Cisco IPS. A case study of some existing signatures will further extend that understanding.

Cisco IPS Signature: Null Byte In HTTP Request (5170/0)

The first existing signature to examine is 5170/0 (Null Byte In HTTP Request). This signature is designed as a generic method of flagging an HTTP request as possibly malicious. It is based on a vulnerability class rather than a specific vulnerability.

In some web applications, input data will be URI escaped; any characters that take the form *%{hex code}* will be translated into the equivalent literal

character. For example, the escape code *%20* will translate into a literal space character.

A NULL byte injection attack takes advantage of escape code translation. A poorly written web application may not account for the way in which translation changes the NULL byte. In the URI-encoded form, a NULL byte is simply the string *%00* and has no effect on the string. However, after an application unescapes the string to its original form, *\x00*, it can be treated by low-level applications as a string termination character.

Signature 5170/0 was created to help mitigate these NULL byte termination attacks. This signature was written using the Service HTTP engine because the signature examines

- *to-service* traffic to a web server
- The URI

This signature uses the *specify-uri-regex* parameter to supply a regular expression that can be matched against the URI. The regular expression in the signature is *\%00*.

The service HTTP engine also has a *de-obfuscate* option. With de-obfuscation turned on, URI-encoded data will be converted back to literal values before the regular expression is matched. However, this option is not turned off for this signature because the original (undecoded) buffer is also tested with the regular expression. This means the signature will also catch double decoding problems.

Cisco IPS Signature: phpMyAdmin PHP Code Injection Vulnerability (26040/0)

The next signature to examine is 26040/0. Rather than being a generic signature to catch a vulnerability class, this signature is more specific to a particular vulnerability: CVE-2009-1151.

This vulnerability is in the phpMyAdmin PHP administration system. After the application is installed, the *setup.php* script remains on the system. This script has permissions that allow anyone to create and execute a PHP script on the web server.

To detect this vulnerability over the network, the signature developer must first isolate the problem to its base elements. This analysis will show that an attacker must execute the script on the web server in the directory */scripts/*, and that the name of the script is *setup.php*.

To detect this behavior, the signature can use the Service HTTP engine and, in the *specify-uri-regex* field, use the regex *[Ss][Cc][Rr][Ii][Pp][Tt][Ss][/][Ss][Ee][Tt][Uu][Pp][.][Pp][Hh][Pp]*. This is exactly what Cisco IPS signature 26040/0 does. However, this behavior by itself would cause the IPS to generate a false-positive alert when this application is set up correctly. Because of this fact, additional criteria are needed to detect this vulnerability.

The code injection vulnerability exists when the application parses the configuration HTTP variable. This information leads to the second criterion. To ensure sure this variable is being inspected, the *specify-arg-name-regex* parameter is used. To make sure the signature catches any particular case, it can use the following regular expression *[cC][Oo][Nn][Ff][Ii][Gg][Uu][Rr][Aa][Tt][Ii][Oo][Nn]*.

However, this new criterion is also not specific enough and would fire on legitimate uses of the setup script. The final step to add to this signature is detection of the actual characters that trigger the vulnerability. The characters that are used in public exploits for this vulnerability are *"Host']="* followed by the payload that is to be injected into the configuration file. This input will break out of the data definition in the file and cause the payload to execute.

To trigger on this information, the *specify-arg-value-regex* parameter is used, which ensures that the signature detects injection of the malicious payload into the *CONFIGURATION* variable and nothing else. For this signature, the regular expression *\x22[Hh][Oo][Ss][Tt]\x27\x5d[=]\x27\x27* is used.

The details about this signature show that Cisco IPS engines provide a precise way to stipulate exactly which traffic is to be inspected by the regular expression, which reduces the chance of false positives and provides a more accurate signature base.

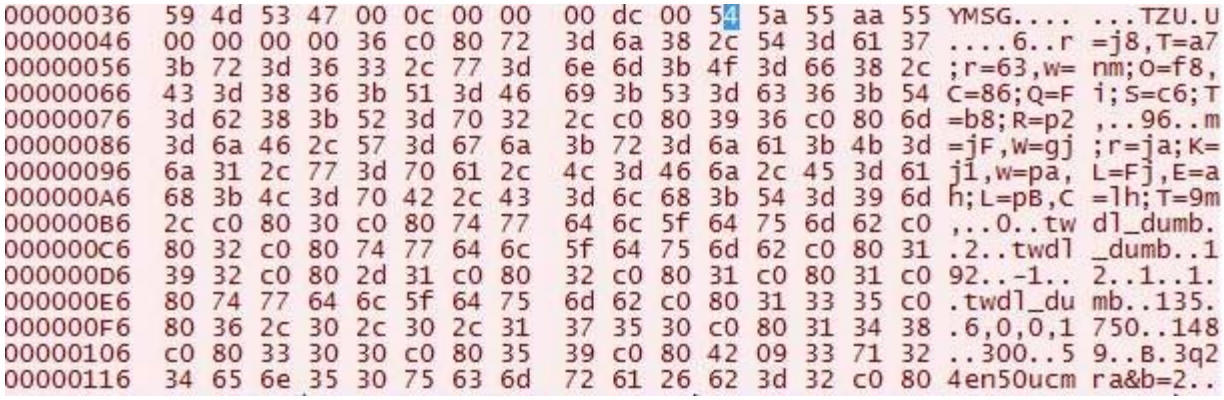
Custom Signatures to Detect Yahoo! Messenger Activity

Observation of the network traffic during the login process of a Yahoo! Messenger client indicates that the stream always starts with *YMSG*, followed by 2 bytes for version, followed by 2 random bytes and then 2 bytes for packet length. The subsequent 2 bytes indicate activity. The information from these 2 bytes, along with the service ports, indicates the type of activity.

Detect Yahoo! Messenger Login

The Yahoo! Messenger client speaks to the Yahoo! server from a random high port to port 5050. The Yahoo! Messenger client login has a challenge-response sequence. The connection starts to/from port 5050, and the 2 bytes that indicate this activity are *\x00\x4c*, which are *service verify*. This is followed by a response with service bytes *\x00\x57*, which are the service authorization. After successful negotiation, the client sends service bytes *\x00\x054* to port 5050. Use the String TCP engine to detect this activity.

The network packet trace of the login activity is illustrated below:



```
00000036  59 4d 53 47 00 0c 00 00  00 dc 00 54 5a 55 aa 55 YMSG.... ...TZU.U
00000046  00 00 00 00 36 c0 80 72  3d 6a 38 2c 54 3d 61 37 ....6..r =j8,T=a7
00000056  3b 72 3d 36 33 2c 77 3d  6e 6d 3b 4f 3d 66 38 2c ;r=63,w= nm;O=f8,
00000066  43 3d 38 36 3b 51 3d 46  69 3b 53 3d 63 36 3b 54 C=86;Q=F i;S=c6;T
00000076  3d 62 38 3b 52 3d 70 32  2c c0 80 39 36 c0 80 6d =b8;R=p2 ,..96..m
00000086  3d 6a 46 2c 57 3d 67 6a  3b 72 3d 6a 61 3b 4b 3d =jF,W=gj ;r=ja;K=
00000096  6a 31 2c 77 3d 70 61 2c  4c 3d 46 6a 2c 45 3d 61 j1,w=pa, L=Fj,E=a
000000A6  68 3b 4c 3d 70 42 2c 43  3d 6c 68 3b 54 3d 39 6d h;L=pB,C =lh;T=9m
000000B6  2c c0 80 30 c0 80 74 77  64 6c 5f 64 75 6d 62 c0 ,..0..tw dl_dumb.
000000C6  80 32 c0 80 74 77 64 6c  5f 64 75 6d 62 c0 80 31 .2..twdl _dumb..1
000000D6  39 32 c0 80 2d 31 c0 80  32 c0 80 31 c0 80 31 c0 92..-1.. 2..1..1.
000000E6  80 74 77 64 6c 5f 64 75  6d 62 c0 80 31 33 35 c0 .twdl_du mb..135.
000000F6  80 36 2c 30 2c 30 2c 31  37 35 30 c0 80 31 34 38 .6,0,0,1 750..148
00000106  c0 80 33 30 30 c0 80 35  39 c0 80 42 09 33 71 32 ..300..5 9..B.3q2
00000116  34 65 6e 35 30 75 63 6d  72 61 26 62 3d 32 c0 80 4en50ucm ra&b=2..
```

The following regular expression detects this activity:

[Y][M][S][G][\x00-\xFF]{6}\x00\x54

The preceding regular expression detects ASCII characters *Y, M, S,* and *G* followed by any six characters followed by hex *\x00\x054*.

The following signature detects this activity:

```
sig-id: 11217
subsig-id: 0
------------------------------------------------
   alert-severity: informational
   sig-fidelity-rating: 85
   promisc-delta: 15
   sig-description
   ---------------------------------------------
      sig-name: Yahoo Messenger Logon
      sig-string-info: 00 54
      sig-comment: empty
      alert-traits: 0
      release: S139
      sig-creation-date: 20050127
      sig-type: Anomaly
   ---------------------------------------------
   engine
   ---------------------------------------------
      string-tcp
      ------------------------------------------
         event-action: produce-alert
         strip-telnet-options: false
         specify-min-match-length
         ---------------------------------------
            no
            ------------------------------------
            ------------------------------------
         ---------------------------------------
         regex-string: [Y][M][S][G][\x00-\xFF]{6}\x00\x54
         service-ports: 5050-5050
         direction: to-service
         specify-exact-match-offset
         ---------------------------------------
            no
            ------------------------------------
               specify-max-match-offset
               ---------------------------------
                  no
                  ------------------------------
                  ------------------------------
               ---------------------------------
               specify-min-match-offset
               ---------------------------------
                  no
                  ------------------------------
                  ------------------------------
               ---------------------------------
            ------------------------------------
         ---------------------------------------
         swap-attacker-victim: false
      ------------------------------------------
   ---------------------------------------------
   event-counter
```

```
      ------------------------------------------------
         event-count: 1
         event-count-key: Axxx
         specify-alert-interval
         ------------------------------------------------
            no
            ------------------------------------------------
            ------------------------------------------------
         ------------------------------------------------
      ------------------------------------------------
      alert-frequency
      ------------------------------------------------
         summary-mode
         ------------------------------------------------
            summarize
            ------------------------------------------------
               summary-interval: 15
               summary-key: Axxx
               specify-global-summary-threshold
               ------------------------------------------------
                  no
                  ------------------------------------------------
                  ------------------------------------------------
               ------------------------------------------------
            ------------------------------------------------
         ------------------------------------------------
      ------------------------------------------------
      status
      ------------------------------------------------
         enabled: false
         retired: false
         obsoletes (min: 0, max: 65535, current: 0)
         ------------------------------------------------
         ------------------------------------------------
      ------------------------------------------------
      vulnerable-os: general-os
      specify-mars-category
      ------------------------------------------------
         yes
         ------------------------------------------------
            mars-category: Info/UncommonTraffic/Chat
         ------------------------------------------------
      ------------------------------------------------
   ------------------------------------------------
```

Detect Yahoo! Messenger Send and Receive Message
The following illustrates network activity when a *send* message is initiated by a Yahoo! Messenger client:

```
000001EA  59 4d 53 47 00 0c 00 00  00 5c 00 06 5a 55 aa 56 YMSG.... .\..ZU.V
000001FA  b5 95 d9 b3 31 c0 80 74  77 64 6c 5f 64 75 6d 62 ....1..t wdl_dumb
0000020A  c0 80 35 c0 80 77 73 75  6c 79 6d c0 80 31 34 c0 ..5..wsu lym..14.
0000021A  80 73 65 6e 64 69 6e 67  20 61 20 74 65 73 74 20 .sending  a test
0000022A  6d 65 73 73 61 67 65 c0  80 39 37 c0 80 31 c0 80 message. .97..1..
0000023A  36 33 c0 80 3b 30 c0 80  36 34 c0 80 30 c0 80 31 63..;0.. 64..0..1
0000024A  30 30 32 c0 80 31 c0 80  32 30 36 c0 80 30 c0 80 002..1.. 206..0..
0000025A  59 4d 53 47 00 0c 00 00  00 62 00 15 00 00 00 00 YMSG.... .b......
0000026A  b5 95 d9 b3 32 31 31 c0  80 4d 73 67 3a 54 6f 74 ....211. .Msg:Tot
0000027A  61 6c 3d 31 09 44 69 73  70 6c 61 79 3d 31 2c 30 al=1.Dis play=1,0
0000028A  2c 30 0a 54 61 62 3a 79  6d 73 67 72 5f 6c 61 75 ,0.Tab:y msgr_lau
0000029A  6e 63 68 63 61 73 74 3d  30 2c 32 36 0a 49 4d 56 nchcast= 0,26.IMV
000002AA  3a 4e 6f 6e 65 3d 31 2c  30 2c 30 2c 30 2c 30 2c :None=1, 0,0,0,0,
000002BA  30 2c 30 0a c0 80 31 30  30 35 c0 80 33 34 35 34 0,0...10 05..3454
000002CA  39 37 35 32 c0 80                               9752..
```

The stream starts with *YMSG* followed by six random characters followed by service bytes *\x00\x06*.

The following regular expression detects this activity:
[Y][M][S][G][\x00-\xFF]{6}\x00\x06

The only difference between detecting a *send* message and a *receive* message is the direction of the service ports. Send message traffic is *to-service*, whereas receive message traffic is *from-service*.

The following signature detects Yahoo! Messenger *send* activity:

```
      sig-id: 11218
```

```
subsig-id: 0
-----------------------------------------------
   alert-severity: informational
   sig-fidelity-rating: 85
   promisc-delta: 15
   sig-description
   -------------------------------------------------
      sig-name: Yahoo Messenger Send Message
      sig-string-info: 00 06
      sig-comment: empty
      alert-traits: 0
      release: S139
      sig-creation-date: 20050127
      sig-type: Anomaly
   -------------------------------------------------
   engine
   -------------------------------------------------
      string-tcp
      -----------------------------------------------
         event-action: produce-alert
         strip-telnet-options: false
         specify-min-match-length
         -------------------------------------------
            no
            -------------------------------------------
            -------------------------------------------
         -------------------------------------------
         regex-string: [Y][M][S][G][\x00-\xFF]{6}\x00\x06
         service-ports: 5050-5050,5101-5101
         direction: to-service
         specify-exact-match-offset
         -------------------------------------------
            no
            -------------------------------------------
               specify-max-match-offset
               -------------------------------------------
                  no
                  -------------------------------------------
                  -------------------------------------------
               -------------------------------------------
               specify-min-match-offset
               -------------------------------------------
                  no
                  -------------------------------------------
                  -------------------------------------------
               -------------------------------------------
            -------------------------------------------
         -------------------------------------------
         swap-attacker-victim: false
      -------------------------------------------------
   -------------------------------------------------
   event-counter
   -------------------------------------------------
      event-count: 1
      event-count-key: Axxx
      specify-alert-interval
      -------------------------------------------------
         no
         -------------------------------------------------
         -------------------------------------------------
      -------------------------------------------------
   -------------------------------------------------
   alert-frequency
   -------------------------------------------------
      summary-mode
      -------------------------------------------------
         summarize
         -------------------------------------------------
            summary-interval: 15
            summary-key: Axxx
            specify-global-summary-threshold
            -------------------------------------------------
               no
               -------------------------------------------------
               -------------------------------------------------
            -------------------------------------------------
         -------------------------------------------------
      -------------------------------------------------
   -------------------------------------------------
   status
   -------------------------------------------------
```

```
            enabled: false
            retired: false
            obsoletes (min: 0, max: 65535, current: 0)
            ------------------------------------------------
            ------------------------------------------------
         ---------------------------------------------
         vulnerable-os: general-os
         specify-mars-category
         ---------------------------------------------
            yes
            ------------------------------------------------
               mars-category: Info/UncommonTraffic/Chat
            ------------------------------------------------
         ---------------------------------------------
      ---------------------------------------------
```

Snort Comparison

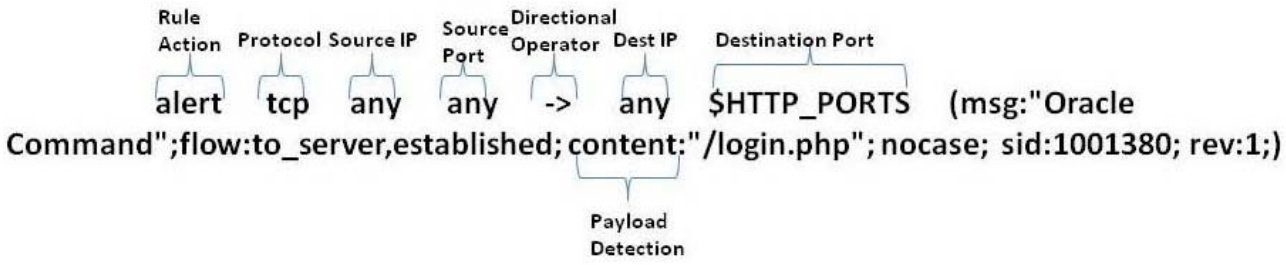
Introduction

Because of the amount of information available, wide deployment, and open source nature of the Snort community, there has been an ongoing need to convert Snort signatures to be used with Cisco IPS. This section provides an overview of the conversion process.

Users are advised that many of the signatures widely available on the Internet have been developed by administrators who have limited knowledge of IPS signature development. Therefore, these signatures may detect a particular exploit instead of detecting all exploits. They may also consume high resources and need to be continually updated to keep pace with changing exploits. Such signatures should be used with caution.

The following figure is an example of a Snort rule:

Figure 13. Snort Rule Example



A Snort signature consists of the following sections:

Rule actions: The rule action tells Snort what to do when it finds a packet that matches the rule criteria. There are five available actions in Snort: alert, log, pass, activate, and dynamic. In inline mode, additional options such as *drop*, *reject*, and *sdrop* are available.

Protocols: Snort supports the following protocols: TCP, UDP, ICMP, and IP.

Directional operator: The directional operator is used to indicate the direction of the traffic to which the Snort rule applies.

Rule options: The following table illustrates Snort rule options and their equivalent Cisco IPS signature parameters.

Table 4. Rule Options

Snort Rule Option	Equivalent Cisco IPS Signature Parameter	Engine
depth	Specify Max Match Offset	String TCP and Atomic IP
dsize	Specify Payload Length	Atomic IP
flags	TCP Flags and TCP Mask	Atomic IP
http_header	Specify HTTP Header Regex	Service HTTP
http_uri	Specify URI Regex	Service HTTP

icmp_id	Specify ICMP Identifier	Atomic IP
icmp_seq	Specify Sequence	Atomic IP
icode	Specify ICMP Code	Atomic IP
itype	Specify ICMP Type	Atomic IP
logto	Log Victim/Attacker/Pair Packets	All engines
msg	Signature Name	All engines
offset	Specify Min Match Offset	String TCP and Atomic IP
pcre	Specify Regex Inspection	String TCP and Atomic IP

Content and Perl-Compatible Regular Expressions vs. Cisco IPS Regular Expressions
The **content** keyword in Snort is used for text and binary content matching. Binary content matching is enclosed in the pipe (|) character. Perl-compatible regular expressions (PCRE) support regular expression matching. The key difference between binary content matching and PCRE is the ability to specify offsets when using the **content** parameter. Offsets cannot be applied to the PCRE section of a Snort signature. Cisco signatures rely exclusively on regexes for matching ASCII and hex characters, and the signatures use and apply offsets to these matches. Offsets play an important role in reducing false positives. Regular expression support in Cisco IPS signatures is limited to the keywords specified at the following link:
http://www.cisco.com/en/US/docs/security/ips/7.0/configuration/guide/idm/idm_signature_engines.html#wp1408334

Depth/Offset vs. Min/Max Match Offset Length
For Cisco IPS, **Min Match Offset** is used to specify the minimum offset into the packet or stream the regular expression must match. **Max Match Offset** is used to specify the maximum offset the regular expression must match. These offsets are very similar to the **offset** and **depth** keywords in Snort, respectively. However, Snort **depth** and **offset** are content modifiers and do not apply to PCRE payload detection matches. The min/max offsets used in Cisco IPS engines apply to regular expression matches also. This makes the process of developing vulnerability-based signatures easier.

Examples
The following examples illustrate the process of converting Snort signatures to Cisco signatures.

Example 1: TCP Streams
The following example illustrates a signature that triggers on established TCP streams.

Snort Signature

The following Snort rule fires on established TCP streams with a destination port of 1024.

alert tcp $EXTERNAL_NET any -> $HOME_NET 1024: (msg:"SPYWARE-PUT Screen-Scraper farsighter runtime detection - initial connection"; flow:to_server,established; content:"|00 00 05 00 00 00|"; depth:8; offset:2; nocase; reference:url,www.spywareguide.com/product_show.php?id=587; classtype:successful-recon-limited; sid:5771; rev:3)

Cisco IPS Custom Signature

An equivalent signature that uses Cisco IPS would use the String TCP engine.

Signature Name: Screen-Scraper farsighter runtime detection
Engine: String TCP
Service Ports: 1024
Direction: To-Service
Regex:\x00\x00\x05\x00\x00\x00
Specify Min Match Offset:Yes
Min Match Offset:2
Specify Max Match Offset:Yes
Max Match Offset:8

Event Action: Produce Alert

Example 2: UDP Packets
The following example illustrates a signature that triggers on UDP packets.

Snort Signature

In the following example, the connection flow type is not specified in the Snort rule. The global configuration is loaded from the Snort configuration file. The default setting is to track sessions for UDP. Explicitly specifying **stateless** as a flow parameter will cause this signature to be packet based rather than stream based. An equivalent Cisco IPS custom signature, illustrated in the following section, uses the Atomic IP engine.

alert udp $EXTERNAL_NET any -> $HOME_NET 15164 (msg:" Keylogger stealthwatcher 2000 runtime detection - agent status monitoring"; content:"|0A 02 08 FE 00|"; depth:10; offset:4; threshold:type limit, track by_src, count 1, seconds 300; metadata:policy security-ips drop; reference:url,www.spywareguide.com/product_show.php? id=879;classtype:successful-recon-limited; sid:6385; rev:2)

Cisco IPS Custom Signature

Signature Name: Keylogger stealthwatcher 2000 runtime detection
Engine: Atomic IP
Specify Layer 4 Protocol: Yes
Layer 4 Protocol: UDP
Specify Payload detection: Yes
Regex String: \x0A\x02\x08\xFE\x00
Specify Min Match Offset: Yes
Min Match Offset: 4
Specify Max Match Offset: Yes
Max Match Offset: 10

Example 3: Distance vs. Minimum Spacing
The **distance** content modifier specifies how far into the packet or stream Snort should start the pattern match relative to the end of the previous pattern match. Similarly, **Minimum Spacing** in the Multi String engine in Cisco IPS specifies the minimum length of spacing between the previous multistring component match. The following examples show how Snort and Cisco IPS use these settings:

Snort Signature

alert tcp $EXTERNAL_NET any -> $HOME_NET 15164 (msg:"Test for MultiString"; flow:to_server,established; content:"|0A 0A 0A 0A|"; content:"|0B 0B 0B 0B|" ;distance:8; metadata:policy security-ips drop; sid:63851; rev:2)

Cisco IPS Custom Signature

Engine: MultiString
Protocol: TCP
Port Selection: Destination
Dest Ports: 15164
Regex Component:
Regex String 1:
\x0A\x0A\x0A\x0A
Regex String 2:
\x0B\x0B\x0B\x0B
Spacing Type: Minimum
Minimum Spacing:8

Example 4: HTTP Header
The following example illustrates signatures that detect attacks based on the HTTP protocol.

Snort Signature

alert tcp any any -> any $HTTP_PORTS (msg:"Test for HTTP Signature"; flow:to_server,established; content:"ABC"; content: "EFG"; http_client_body;content:"DEF";http_header;content:"JLM";http_uri)

Cisco IPS Custom Signature

Engine:Service HTTP
Service Ports: #WEBPORTS
Specify Request Regex: yes
Request Regex: EFG
Specify Header Regex: Yes
Header Regex: DEF
Specify URI Regex: Yes
URI Regex: JLM

Example 5: Non-Payload Detection Rule Options
The following table identifies Snort non-payload detection rule options.

Table 5. Snort Non-Payload Detection Rule Options

Option	Description
ttl	Checks the IP TTL value
tos	Checks the IP TOS field for a specific value
fragbits	Checks whether fragmentation and reserved bits are set in the IP header
flags	Checks whether specific TCP flag bits are present

Snort Signature

alert tcp any any -> any any (msg"IP Options test";ttl:3;tos:4;fragbits:M;flags:SF,12)

Cisco IPS Custom Signature

Engine: Atomic IP
Fragment Status: Fragmented
Specify IP Time-To-Live: Yes
TTL: 3
Specify IP Type-of-Service:
IP Type of Service:4
Specify Layer 4 Protocol: Yes

Layer 4 Protocol: TCP

TCP Flags: SYN, FIN

TCP Mask: SYN, FIN

Example 6: Flowbits vs. Meta
The **flowbit** parameter is used in Snort to track the state of a transport layer protocol stream. Where Snort uses the **flowbit** parameter along with the *set* and *isset* options, an equivalent Cisco IPS signature will use the Meta engine. The following examples illustrate this process.

Snort Signature

alert tcp any $HTTP_PORTS -> any any (msg:" Signature 1"; flow:to_server,established; content:"abc"; flowbits:set,abc_check; flowbits:noalert;)

alert tcp any $HTTP_PORTS -> any any (msg:" Signature 2"; flow:to_server,established; content:"def"; flowbits:set,def_check; flowbits:noalert;)

alert tcp any $HTTP_PORTS -> any any (msg:" Signature 3"; flow:to_server,established; flowbits:isset,abc_check; flowbits:isset,def_check;)

Cisco IPS Custom Signature

Component 1:
Engine: String-TCP
Signature Name: Signature 1
alert severity: informational

Sig Fidelity Rating: 60
service ports: #WEBPORTS
direction: from service
Event Action: none selected
regex: abc

Component 2:
Engine: String-TCP
Signature Name: Signature 1
alert severity: informational
Sig Fidelity Rating: 60
service ports: #WEBPORTS
direction: from service
Event Action: none selected
regex: def

Meta Signature:
Engine: Meta
Signature Name: Signature 3
event action: produce alert
Component List: Component 1 and Component 2
All Components Required: Yes

Acknowledgments

Martin Zeiser, Software Engineer
Neil Archibald, Software Engineer
Shiva Persaud, Software Engineer
Roopesh Uppala, Software Engineer

Back to Top

Cisco Security Intelligence Operations