

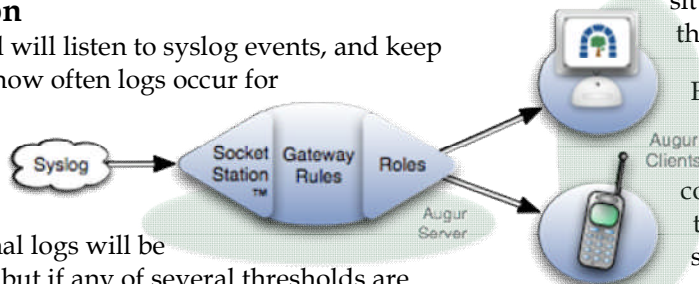
## The Situation:

Your firewall software writes activity logs through your Syslog server. You regularly review the logs, and you occasionally find denials on port 445. Sometimes those attacks are frequent, impacting service. You'd like to know when attacks occur, but you don't want to be bothered about occasional incidents. You've decided that a dozen denials in any 5-minute window are worthy of your attention. Just to be safe, you'd also like to know if 100 events occur spread out across any 1-hour period, or even 200 in any 4-hour period.

## Solution

NetAvail will listen to syslog events, and keep track of how often logs occur for port 445

denials. Occasional logs will be ignored, but if any of several thresholds are exceeded, NetAvail will generate an alert and optional notifications.



## Setup

Live Syslog logs are collected by NetAvail's Socket Station™ module and processed by an NetAvail gateway. We'll configure the gateway to interpret each line as a separate event message. (NetAvail can also handle events that span multiple lines.)

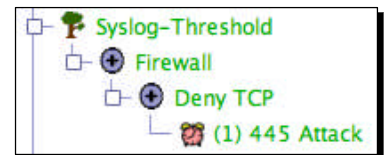
The NetAvail rules then need to identify various log types. For this situation, we're interested in

## NetAvail™ Threshold Alerting

denials to port 445. We'll also want to parse out some variables that can be used later to construct alert text. Then, NetAvail needs to keep track of these events so we can apply various alarm thresholds to their frequency as new logs arrive. It sounds complicated, but NetAvail's configuration GUI is optimized to simplify typical business rules like this.

The rule tree will contain a few nodes. The first ones will identify the log type, and parse our variables along the way. (These nodes will also provide convenient branch points where we can add nodes for processing additional log types later.) The last node we'll need for our current situation will define the alarm thresholds and text.

Each rule node contains a true/false pattern test. If an event matches the pattern, then processing continues along that path in the rule tree. Since many applications may be sending logs to our Syslog, our first node, named **Firewall**, will filter for logs from our firewall. We'll just need to define



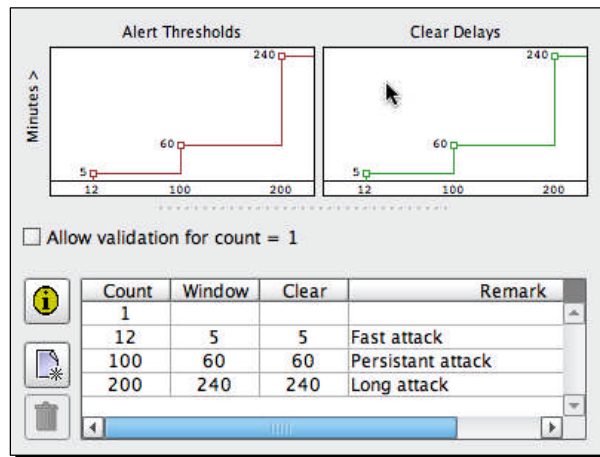
its pattern with the string "ipfw:" to match any logs from your IP firewall. We could get fancy by using *regular expression* patterns, but this simple text pattern should do the trick.

## Syslog:

```
Jan 21 00:46:53 HenryNg ipfw: 12190 Deny TCP 10.9.255.13:2782 10.9.40.87:445 in via ppp0
Jan 21 00:46:54 HenryNg ipfw: 12190 Deny TCP 10.9.254.233:4226 10.9.40.87:445 in via ppp0
Jan 21 00:46:54 HenryNg ipfw: 12190 Deny TCP 10.9.98.233:1670 10.9.40.87:135 in via ppp0
Jan 21 00:46:56 HenryNg ipfw: 12190 Deny TCP 10.9.255.13:2782 10.9.40.87:445 in via ppp0
Jan 21 00:46:57 HenryNg ipfw: 12190 Deny TCP 10.9.254.233:4226 10.9.40.87:445 in via ppp0
Jan 21 00:47:08 HenryNg ipfw: 12190 Deny TCP 10.9.252.166:2012 10.9.40.87:135 in via ppp0
Jan 21 00:47:11 HenryNg ipfw: 12190 Deny TCP 10.9.252.166:2012 10.9.40.87:135 in via ppp0
```

The next node, named **Deny TCP**, will match any firewall TCP denials. Here, we just need to define the pattern "Deny TCP". We can use such a simple pattern because the previous node already filtered out everything except IP firewall logs, and we know the strict format of these logs. Leveraging that knowledge, this would be a good place to parse out the attacker's IP address. We'll use a regular expression for that. The pattern "TCP ([^:]+)" will capture the text between "TCP" and the next colon, which contains the attacker's IP address. We'll store this value in a variable named **attacker**.

Our last node will match only the denial logs for port 445. The simple text pattern ":445 in" will work fine. Since this node will be performing the alert validation, this is where we'll define the



thresholds and the alert text. Thresholding is configured by setting your requirements in a table. Any number thresholds can be defined together. For our scenario we'll define three. The first two columns are the event count and the window of time, in minutes. The third column controls how alerts should be cleared, and should usually be the same as the second field. The last column is an optional remark field.

You'll notice a graph that represents the thresholds defined in the table. This graph is interactive, so if you like, you can click and drag the threshold lines with your mouse; the numbers in the table will automatically be adjusted.

Lastly, we can define the text that would appear in alerts and paging. For the alert's element name, we usually reference variables, but in this

**Compose Alert**

Element:

Summary:

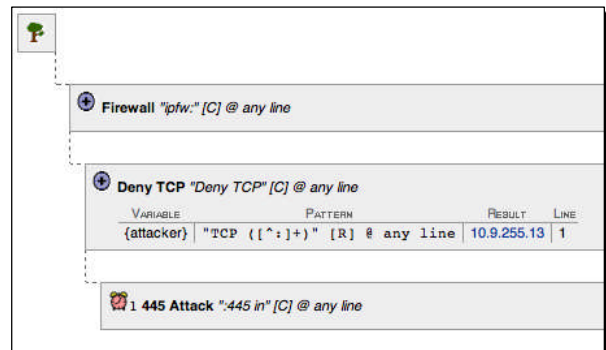
case we'll use some static text. For the alert's summary, we will use both static text and a reference to the **attacker** variable we previously stored.

That's it for the alert rules. Optionally you can subscribe to get paged for this alert too..

## Testing

After configuration, it's a good idea to do some testing before going into production. NetAvail includes a handy trace tool that runs test data through a rule tree, performing the exact same work as a live NetAvail gateway. The trace output shows the path each event travels, any variables parsed out, and the final matching node.

Below you can see the trace results from sending



the first Syslog line through the tool.