

Threshold Alerting

The Situation

Your firewall software writes 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 occur spread out across any 1-hour period, or even 200 in any 4-hour period.

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 10.9.254.22:4226 10.9.40.87:445 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 10.9.254.22:4226 10.9.40.87:445 in via ppp0
```

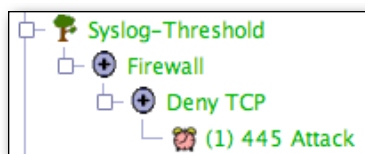
Solution

You will configure to Augur process syslog events, and have it keep track of how often logs occur for port 445 denials. Occasional logs will be ignored, but if any of several thresholds are exceeded, Augur will generate an alert and optional notifications.

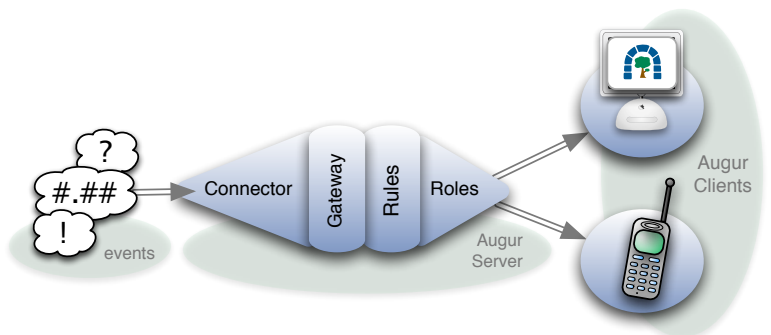
Setup

Augur uses *connector* plug-ins to receive events. We'll configure a *gateway* to use one of the included connectors to receive syslog. The gateway will interpret each line as a separate event message. (Augur can also handle events that span multiple lines.)

An Augur rule tree identifies events by pattern. We're interested in denials to port 445. We also want to parse out some variables that we can use later. Lastly, we want Augur to keep track of these events so we can apply frequency thresholds as new events arrive. It sounds complicated, but Augur's configuration GUI is optimized to help with typical business rules like this.



The rule tree will contain a few nodes. The first ones will identify the log type by successively



narrowing the matching pattern restrictions. At nodes along the way, we will parse out variables. The last node will define the alarm and our thresholds.

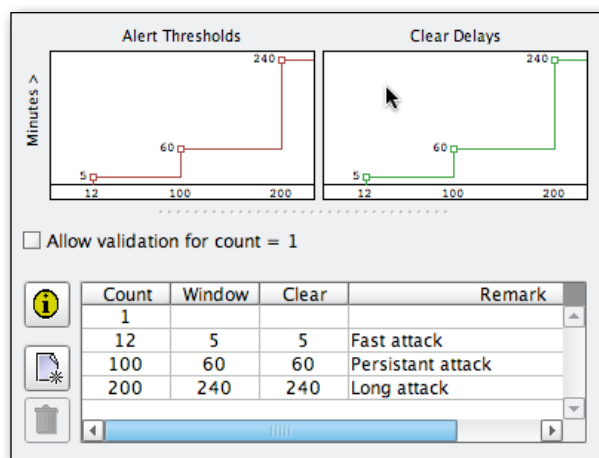
Each rule node contains a true/false pattern test. If an event matches the pattern, then processing continues along that path in the rule tree. Since many applications may be sending logs to our syslog, our first node will match only logs from our firewall. We will name the node, "Firewall".

To define its pattern test, we will use the plain text pattern, ". ipfw:" (The first character is a space. In patterns, Augur always displays spaces as dots, for clarity.) We could use a *regular expression* pattern, but this simple text pattern is enough.

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 parent node already filtered out everything except IP firewall logs.

This node 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. We’ll store this value in a variable named **attacker**.

Our last node will match only the denial logs for port 445. The text pattern “:445·in” will work. Since this node will be performing the alert validation, this is where we’ll define the thresholds and the alert text. You can also define prioritization, time-outs, plug-in customizations and more, but we won’t need to cover those features for this example.



Thresholding is configured by setting your requirements in a table. Any number of 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; it 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 you can click and drag the threshold lines with your mouse. The numbers in the table will automatically update.

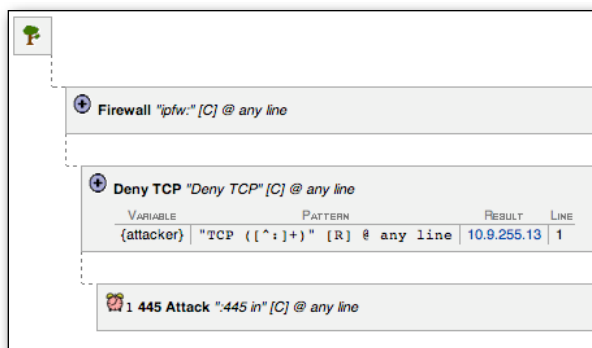
Lastly, we can define the text to appear in the Alert Viewer, and in notifications. For the alert’s element name, we usually need to reference variables, but in this case we can define some static text, as shown below. For the alert’s summary, we will use both static text and a reference to the **attacker** variable we previously stored.

Alert Element: Firewall TCP Port 445
 Alert Summary: Port is under attack from %attacker%.|

That’s it for the alert rules. Later, you may want to subscribe for paging notification on this alert too.

Testing

After all that configuration, you’ll want to do some testing to verify the pattern matching and parsing. Augur includes a trace tool that runs test data through a rule tree, performing the exact same work as a live Augur gateway. The trace output shows the path that each event travels and the results of any variable parsing. Below, you can see the trace results from sending one of our syslog line through the tool.



For testing the thresholds, interactions with other alerts, and more, you can use the **Injector** tool. This tool lets you manually send data through a live testing gateway. You load a sample of your events into the Injector, highlight one or more events, then press the **Send** button. You can view the results in the Alert Viewer. The full procedure is detailed in a separate document.

Now you’re ready to enable the gateway, and let Augur automate your event management.