Pulse Secure®

WHITE PAPER

Traffic Valuation and Prioritization using the Pulse Secure Virtual Traffic Manager

TABLE OF CONTENTS

Introduction

Whether running an e-commerce web site, online corporate services or an internal intranet, there's always the need to squeeze more performance from limited resources and to ensure that the most valuable users get the best possible levels of service from the services hosted. Flash crowd events, greedy or malicious users, and poorly performing applications all affect the level of service you deliver. The powerful traffic management capabilities of Pulse Secure Virtual Traffic Manager (vTM) allows definition of the service policies that deal with these problems, and apply them easily and non-intrusively to all the traffic.

An Example of Traffic Valuation and Prioritization

Imagine running a successful gaming service in a glamorous location. The service is growing daily, and many long-term users are becoming very valuable. Unfortunately, much of the bandwidth and server hits are taken up by competitor robots that screen-scrape betting statistics, and poorly written bots that spam the gaming tables and occasionally place low-value bets.

At certain times of the day, this activity is so great that it impacts the quality of the service delivered, and most valuable customers are affected. Pulse Secure Virtual Traffic Manager's ability to measure, classify, and prioritize traffic, can construct a service policy that comes into effect when the web site begins to run slowly.

The policy can inspect and classify each request, looking at information like request URLs, cookies, and form data, and it can record data in cookies. Having classified the request, the policy then enforces different levels of service:

- Competitor's screen-scraping robots are tightly restricted to one request per second. A 10-second delay reduces the value of the information they screen-scrape.
- Users who have not yet logged in are limited to a small proportion of the available bandwidth and directed to a pair of basic web servers, thus reserving capacity for users who are logged in.
- Users who have made large transactions in the past are tagged with a cookie and the performance they receive is measured. If they are receiving poor levels of service (over 100ms response times), then some of the transaction servers are reserved for these high-value users and the activity of other users is shaped by a system-wide queue.

Whether operating a gaming service, a content portal, a B2B or B2C e-commerce site, or an internal intranet, this kind of service policy can help ensure that key customers get the best possible service, minimize the churn of valuable users, and prevent undesirable visitors from harming the service to the detriment of others.

Designing a Service Policy

This white paper describes techniques to address the following problems:

- Guarantee certain levels of service for certain users.
- Prioritize some transactions over others.
- Restrict the activities of certain users.

To address these problems, an administrator must consider the following questions:

1. Under what circumstances the policy to take effect?
2. How to categorize your users?
3. How to apply the differentiation?

One or more TrafficScript rules can be used to apply the policy. They take advantage of the following Pulse Secure Virtual Traffic Manager features:

4. **When does the policy take effect?**
   - *Service Level Monitoring:* Measure system performance and apply policies only when they are needed.
   - *Custom Logging:* Log and analyze activity to record and validate policy decisions.
5. **How are users categorized?**
   - *Application traffic inspection:* Determine source, user, content, value; XML processing with XPath searches and calculations.
6. **How are they given different levels of service?**
   - *Request Rate Shaping:* Apply fine-grained rate limits for transactions.
   - *Bandwidth Control:* Allocate and reserve bandwidth.
   - *Traffic Routing and Termination:* Route high and low priority traffic differently; terminate undesired requests.
   - *Selective Traffic Optimization:* Selective caching and compression.

Pulse Secure Virtual Traffic Manager TrafficScript Rules

Pulse Secure vTM functions as an application proxy, accepting clients' requests and load-balancing them across a number of back-end servers.

Central to Pulse Secure vTM is the concept of TrafficScript rules. Using the simple TrafficScript programming language, you can write scripts that:

- Inspect all aspects of requests and responses and perform complex calculations such as XSLT transforms.
- Modify request or response data as it passes through the Pulse Secure vTM.
- Fine-tune the behavior of Pulse Secure vTM for each request.

TrafficScript includes a large number of helper functions to facilitate the inspection, transformation, and routing of traffic. These rules are run at two different times:

- Whenever Pulse Secure vTM receives a client request (a request rule);
- Whenever Pulse Secure vTM receives a response from the back-end server (a response rule).

For example, the following TrafficScript request rule inspects HTTP requests. If the request is for a .jsp page, the rule looks at the clients' priority cookie and routes the request to the high-priority or low-priority server pools as appropriate (see Figure 1).

Generally, if describing the traffic management logic that is required, it is possible to implement it using TrafficScript.

This document will refer to TrafficScript rules extensively as it describes how to implement service policies.

```
$url = http.getPath();
if( string.endsWith( $url, ".jsp" ) ) {
   $cookie = http.getCookie( "Priority" );

   if( $cookie == "high" ) {
      pool.use( "high-priority" );
   } else {
      pool.use( "low-priority" );
   }
}
```

Figure 1: Routing a request to a server pool.

Service Level Monitoring

Using Service Level Monitoring, Pulse Secure Virtual Traffic Manager can measure and react to changes in response times for the hosted services by comparing response times to a desired time.

Service Level Monitoring can be configured by creating a Service Level Monitoring Class (SLM Class). The SLM Class is configured with the desired response time (for example, 100ms), and some thresholds that define actions to take. For example, suppose that the performance of our Java servlets is a matter of concern. SLM Class with the desired performance can be configured and used to monitor all requests for Java servlets (see Figure 2).

As shown in Figure 3, the performance figures generted by the Java servlets SLM Class can be monitored to discover the response times and the proportion of requests that fall outside the desired response time. (See Figure 3.) Once requests are monitored by an SLM Class, the proportion of requests that meet the desired response time within a TrafficScript rule can be discovered. This makes it possible to implement TrafficScript logic that is only called when services are underperforming.

Example: Simple Differentiation

Suppose there is a TrafficScript rule that tested to see if a request came from a high-value customer.

- When service is running slowly, high-value customers should be sent to one server pool (gold) and other customers sent to a lower-performing server pool (bronze).
- However, when the service is running at normal speed, all customers should be sent to all servers (the server pool named “all servers”).

The following TrafficScript rule describes how this logic can be implemented (Figure 4).

```
$url = http.getPath();
if( string.startsWith( $url, "/servlet/" ) {
   connection.setServiceLevelClass( "Java servlets" );
}
```

Figure 2: Assigning a service class to a request.

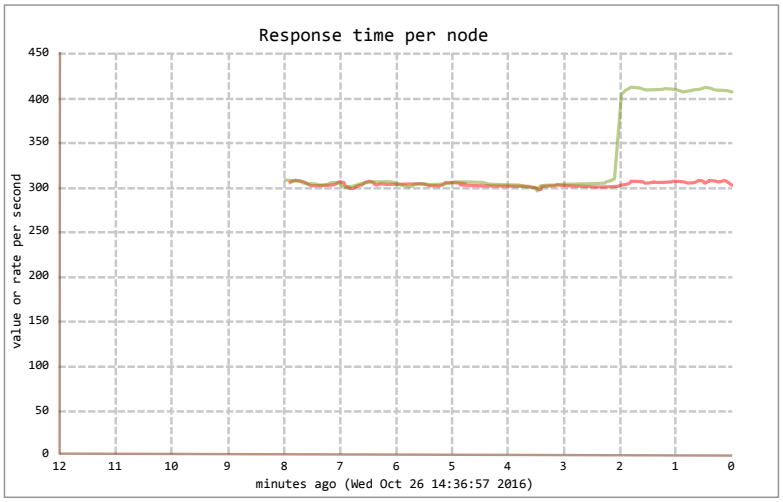

Figure 3: Activity chart showing response time per node.

```
# Monitor all traffic with the 'response time' SLM class, which is
# configured with a desired response time of 200ms

connection.setServiceLevelClass( "response time" );

# Now, check the historical activity (last 10 seconds) to see if it's
# been acceptable (more than 90% requests served within 200ms)

if( slm.conforming( "response time" ) > 90 ) ) {
   # select the 'all servers' server pool and terminate the rule
   pool.use( "all servers" );
}

# If we get here, things are running slowly

# Here, we decide a customer is 'high value' if they have a login cookie,
# so we penalize customers who are not logged in. You can put your own
# test here instead

$logincookie = http.getCookie( "Login" );
if( $logincookie ) {
   pool.use( "gold" );
} else {
   pool.use( "bronze" );
}
```

Figure 4: TrafficScript example to prioritize requests.

Application Traffic Inspection

There is no limit to how traffic can be inspected and valued. Pulse Secure Virtual Traffic Manager's Application Traffic Inspection allows to look at any aspect of a client's request, so that it can then be categorized as needed. (See Figure 5.)

Remembering the Classification with a Cookie

Often, it only takes one request to identify the status of a user, but this decision needs to be remembered for all subsequent requests. For example, if a user places an item in his shopping cart by accessing the URL /cart.php, then it is important to remember this information for all of his subsequent requests.

Adding a response in a Response Rule with the http.setResponseCookie() function as shown in Figure 6.

It can be done in a Request Rule as shown in Figure 7.

This cookie will be sent by the client on every subsequent request, so to test if the user has placed items in his shopping cart, testing for the presence of the GotItems cookie in each request rule is the only thing needed as shown in Figure 8 on the next page.

If necessary, the cookie can be encrypted and signed so that it cannot be spoofed or reused as shown in Figure 9 on the next page.

Request Rate Shaping

Having decided when to apply the service policy (using Service Level Monitoring), and classified the users (using Application Traffic Inspection), now there is the need to decide how to prioritize valuable users and penalize undesirable ones.

```
# What is the client asking for?
$url = http.getPath();

# ... and the QueryString
$qs = http.getQueryString();

# Where has the client come from?
$referrer = http.getHeader( "Referer" );

# What sort of browser is the client using?
$ua = http.getHeader( "User-Agent" );

# Is the client trying to spend more than $49.99?
if( http.getPath() == "/checkout.cgi"
     && http.getFormParam( "total" ) > 4999 ) {
   # Take some action here
   # ....
}

# What's the value of the CustomerName field
# in the XML purchase order # in the SOAP request?

$body = http.getBody();
$name = xml.xpath.matchNodeSet( $body, "", "//Info/CustomerName/text()");

# Take the name, post it to a database server with a web interface and
# inspect the response. Does the response contain the value 'Premium'?

$response = http.request.post( "http://my.database.server/query",
"name=".string.htmlEncode( $name ) );

if( string.Contains( $response, "Premium" ) ) {
   # Take some action here
   # ....
}
```

Figure 5: Inspecting client requests.

```
if( http.getPath() == "/cart.php" ) {
   http.setResponseCookie( "GotItems", "Yes" );
}
```

Figure 6: Setting cookies in a Response Rule.

```
if( http.getPath() == "/cart.php" ) {
   http.setResponseHeader( "Set-Cookie", "GotItems=Yes" );
}
```

Figure 7: Setting cookies in a Request Rule.

Pulse Secure Virtual Traffic Manager's Request Rate Shaping capability is used to apply maximum request rates:

- On a global basis ("no more than 100 requests per second to my application servers");
- On a very fine-grained per-user or per-class basis ("no user can make more than 10 requests per minute to any of my statistics pages").

A service policy that places limits on a wide range of events can be constructed, with very fine-grained control over how events are identified. Per-second and per-minute rates on these events can be imposed.

For example:

- Rate-shape individual web spiders to stop them from overwhelming your web site. Each web spider, from each remote IP address, can be given maximum request rates.
- Throttle individual SMTP connections, or groups of connections from the same client, so that each connection is limited to a maximum number of sent emails per minute.
- Rate-shape new SMTP connections so that a remote client can only establish new connections at a particular rate.
- Apply a global rate shape to the number of connections per second that are forwarded to an application.
- Identify individual user's attempts to log in to a service, and then impede any dictionary-based login attacks by restricting each user to a limited number of attempts per minute.

Request Rate Limits are imposed using the TrafficScript `rate.use()` function, and can be configured per-second and per-minute limits in the rate class. Both limits are applied (note that if the per-minute limit is more than 60-times the per-second limit, it has no effect).

```
if( http.getCookie( "GotItems" ) ) {
...
}
```

Figure 8: Testing to see whether a cookie has been set.

```
# Setting the cookie

# Create an encryption key using the client's IP address and user agent
# Encrypt the current time using encryption key; it can only be decrypted
# using the same key

$key = http.getHeader( "User-Agent" ) . ":" . request.getRemoteIP();
$encrypted = string.encrypt( sys.time(), $key );
$encoded = string.hexencode( $encrypted );
http.setResponseHeader( "Set-Cookie", "GotItems=".$encoded );

# Validating the cookie
$isValid = 0;
if( $cookie = http.getCookie( "GotItems" ) ) {

   $encrypted = string.hexdecode( $cookie );

   $key = http.getHeader( "User-Agent" ) . ":" . request.getRemoteIP();
   $secret = string.decrypt( $encrypted, $key );

   # If the cookie has been tampered with, or the ip address or user
   # agent differ, the string.decrypt will return an empty string.
   # If it worked and the data was less than 1 hour old, it's valid:
   if( $secret && sys.time()-$secret < 3600 ) {
   $isValid = 1;
   }
}
```

Figure 9: Using an encrypted cookie.

Using a Rate Class

Rate classes function as queues. When the TrafficScript `rate.use()` function is called, the connection is suspended and added to the queue that the rate class manages. Connections are then released from the queue according to the per-second and per-minute limits.

There is no limit to the size of the backlog of queued connections. For example, if 1000 requests arrived in quick succession to a rate class that admitted 10 per second, 990 of them would be immediately queued. Each second, 10 more requests would be released from the front of the queue.

While they are queued, connections may time out or be closed by the remote client. If this happens, they are immediately discarded.

The `rate.getBacklog()` function can be used to discover how many requests are currently queued. If the backlog is too large, administrators may decide to return an error page to the user rather than risk their connection timing out. For example, to rate shape jsp requests, but defer requests when the backlog gets too large (Figure 10).

Rate Classes with Keys

In many circumstances, more fine-grained rate-shape limits may be applied. For example, in a login page there is the need to limit how frequently each individual user can attempt to log in to just 2 attempts per minute.

The `rate.use()` function can take an optional key which identifies a specific instance of the rate class. This key can be used to create multiple, independent rate classes that share the same limits, but enforce them independently for each individual key.

For example, the login limit class is restricted to two requests per minute to limit how often each user can attempt to log in (Figure 11).

This rule can help to defeat dictionary attacks where attackers try to brute-force crack a user's password. The rate shaping limits are applied independently to each different value of $user. As each new user accesses the system, they are limited to two requests per minute, independently of all other users who share the "login limit" rate shaping class.

Applying service policies with rate shaping

Of course, once the users are classified, different rate settings to different categories of users can be applied (Figure 12).

If the service is running slowly, rate-shape users who have not placed items into their shopping cart with a global limit, and rate-shape other users to eight requests per second each (Figure 13 on the next page).

Bandwidth Shaping

Bandwidth shaping allows Pulse Secure Virtual Traffic Manager to limit the number of bytes per second used by inbound or outbound traffic, for an entire service, or by the type of request.

```
$url = http.getPath();

if( string.endsWith( $url, ".jsp" ) ) {
   if( rate.getBacklog( "shape requests" ) > 100 ) {
      http.redirect( "http://mysite/too_busy.html" );
   } else {
      rate.use( "shape requests" );
   }
}
```

Figure 10: Using a rate class to manage web site congestion.

```
$url = http.getPath();
if( string.endsWith( $url, "login.cgi" ) ) {
   $user = http.getFormParam( "username" );
   rate.use( "login limit", $user );
}
```

Figure 11: Using a rate class to restrict repeated login requests.

```
# If they have an odd-looking user agent, or if there's no host header,
# the client is probably a web spider. Limit it to 1 request per second.
$ua = http.getHeader( "User-Agent" );

if(  ! string.startsWith( $ua, "Mozilla/" ) &&
     ! string.startsWith( $ua, "Opera/" ) ||
   ! http.getHeader( "Host" ) ) {
  rate.use( "spiders", request.getRemoteIP() );
}
```

Figure 12: Using a rate class to restrict automated Web scans and spiders.

Bandwidth limits are automatically shared and enforced across all the Pulse Secure vTM machines in a cluster. Individual Pulse Secure vTM machines take different proportions of the total limit, depending on the load on each, and unused bandwidth is equitably allocated across the cluster depending on the need of each machine.

Like Request Rate Shaping, administrators can use Bandwidth shaping to limit the activities of subsets of users. For example, there can be a 20Mbs network connection that is being over-utilized by a certain type of client, and which is affecting the responsiveness of the service. There can be therefore the need to limit the bandwidth available to those clients to 2Mbs.

```
if( slm.conforming( "timer" ) < 80 ) {
   $cookie = request.getCookie( "Cart" );
   if( !       $cookie ) {
   rate.use( "casual users" );
   } else {
      # Get a unique id for the user
      $cookie = request.getCookie( "JSPSESSIONID" );
      rate.use( "8 per second", $cookie );
   }
}
```

Figure 13: Using a rate class to prioritize users who have items in their shopping cart.

```
response.setBandwidthClass( "class name" );
```

Figure 14: Using a bandwidth class to control overall network usage.

Using Bandwidth Shaping

Like Request Rate Shaping, a Bandwidth class with a maximum bandwidth limit can be configured. Connections are allocated to a class as follows (Figure 14).

All of the connections allocated to the class share the same bandwidth limit.

Example: Managing Flash Floods

The following example helps to mitigate the Slashdot Effect, a common example of a Flash Flood problem. In this situation, a web site is overwhelmed by traffic as a result of a high-profile link (for example, from the Slashdot news site), and the level of service that regular users experience suffers as a result. (See Figure 15.) The example looks at the Referer header, which identifies where a user has come from to access a web site. If the user has come from slashdot.org, he is tagged with a cookie so that all of his subsequent requests can be identified, and he is allocated to a low-bandwidth class (Figure 16).

Figure 15: Example of managing flash floods.

Traffic Routing and Termination

Different levels of service can be provided by different traffic routing, or in extreme events, by dropping some requests.

For example, some large media sites provide different levels of content; high-bandwidth rich media versions of news stories are served during normal usage, and low-bandwidth versions are served when traffic levels are extremely high. Many websites provide flash-enabled and simple HTML versions of their home page and navigation.

This is also commonplace when presenting content to a range of browsing devices with different capabilities and bandwidth.

The switch between high and low bandwidth versions could take place as part of a service policy: as the service begins to under-perform, some (or all) users could be forced onto the low-bandwidth versions so that a better level of service is maintained. (See Figure 17.)

Example: Ticket Booking Systems

Ticket Booking systems for major events often suffer enormous floods of demand when tickets become available.

Pulse Secure Virtual Traffic Manager's request rate shaping system can be used to limit how many visitors are admitted to the service, and if the service becomes overwhelmed, 'please try again' message can be sent back rather than keeping the user 'on hold' in the queue indefinitely.

Suppose the 'booking' rate shaping class is configured to admit 10 users per second, and that users enter the booking process by accessing the URL "/bookevent?eventID=<id>". This rule ensures that no user is queued for more than 30 seconds, by keeping the queue length to no more than 300 users (10 users/second * 30 seconds) (Figure 18).

Example: Prioritizing Resource Usage

In many cases, the resources are limited and when a site is overwhelmed, users' requests still need to be served.

Consider the following scenario:

- The site runs a cluster of four identical application servers (servers one to four);
- Users are categorized into casual visitors and customers; customers have a 'Cart' cookie, and casual visitors do not.

```
$referrer = http.getHeader( "Referer" );
if( string.contains( $referrer, "slashdot.org" ) ) {
   http.addResponseHeader(
      "Set-Cookie", "slashdot=1" );
   connection.setBandwidthClass( "slashdot" );
}

if( http.getCookie( "slashdot" ) ) {
   connection.setBandwidthClass( "slashdot" );
}
```

Figure 16: Using a bandwidth class to manage a flash flood from a referring Web site.

```
# Forcibly change requests that begin /high/ to /low/
$url = http.getPath();
if( string.startsWith( $url, "/high" ) ) {
   $url = string.replace( $url, "/high", "/low" );
   http.setPath( $low );
}
```

Figure 17: Managing traffic congestion by routing requests to a lower bandwidth service.

```
# limit how users can book events

$url = http.getPath();
if( $url == "/bookevent" ) {
   # How many users are already queued?
   if( rate.getBacklog( "booking" ) > 300 ) {
       http.redirect( "http://www.mysite.com/too_busy.html");
   } else {
      rate.use( "booking" );
   }
}
```

Figure 18: Using a rate class to implement a simple queue for a ticket booking system.

The goal is to give all users the best possible level of service, but if customers begin to get a poor level of service, they need to be prioritized over casual visitors. More then 80 percent of customers need to get responses within 100ms.

This can be achieved by splitting the four servers into two pools: the 'allservers' pool contains servers one to four, and the 'someservers' pool contains servers one and two only.

When the service is poor for the customers, the casual visitors will be restricted to just the some servers pool. This effectively reserves the additional servers three and four for the customers' exclusive use. The following code uses the 'response' SLM class to measure the level of service that customers receive (Figure 19).

Selective Traffic Optimization

Some of the features in Pulse Secure Virtual Traffic Manager can be used to improve the end user's experience, but they take up resources on the Pulse Secure vTM system:

- Content Compression reduces the bandwidth used in responses and gives better response times, but it takes considerable CPU resources and can degrade performance.
- Content Caching can give much faster responses, and it is possible to cache multiple versions of content for each user. However, this consumes memory on the Pulse Secure vTM system.

Both of these features can be enabled and disabled on a per-user basis, as part of a service policy.

Content Compression

Use the http.compress.enable() and `http.compress.disable()` TrafficScript functions to control whether or not Pulse Secure vTM will compress response content to the remote client.

Note that Pulse Secure vTM will only compress content if the remote browser has indicated that it supports compression.

On a lightly loaded system, it's appropriate to compress all response content whenever possible (Figure 20).

On a Pulse Secure Virtual Traffic Manager where the CPU usage is becoming too high, administrators can selectively compress content (Figure 21).

Content Caching

Pulse Secure Virtual Traffic Manager can cache multiple different versions of a HTTP response. For example, if the home page is generated by an application that customizes it for each user, Pulse Secure Virtual Traffic Manager can cache each version separately, and return the correct version from the cache for each user who accesses the page.

```
$customer = http.getCookie( "Cart" ); if( $customer ) {
connection.setServiceLevelClass( "response" );
pool.use( "allservers" );
} else {
   if( slm.conforming( "response" ) < 80 ) {
      pool.use( "someservers" );
   } else {
      pool.use( "allservers" );
   }
}
```

Figure 19: Assigning more resources to users who have items in their shopping cart.

```
http.compress.enable();
```

Figure 20: Optimizing bandwidth by compressing client responses.

```
# Don't compress by default
http.compress.disable();

if( $isvaluable ) {
   # do compress in this case
   http.compress.enable();
}
```

Figure 21: Selective compression lets you decide when to optimize only some types of content.

Pulse Secure vTM's cache has a limited size so that it does not consume too much memory and cause performance to degrade. Administrators may wish to prioritize which pages you put in the cache, using the `http.cache.disable()` and `http.cache.enable()` TrafficScript functions.

Note: Content Caching in your Pulse Secure vTM Virtual Server configuration also needs to be enabled; otherwise the TrafficScript cache control functions will have no effect (Figure 22).

Custom Logging

A service policy can be complicated to construct and implement.

The TrafficScript functions `log.info()`, `log.warn()` and `log.error()` are used to write messages to the Pulse Secure vTM event log, and so are very useful debugging tools to assist in developing complex TrafficScript rules.

For example, the code in Figure 23 will append the message in Figure 24 to your error log file.

Error log file can be inspected by viewing the 'Event Log' on the Pulse Secure vTM Admin Server. When debugging a rule, use log.info()can be used to print out progress messages as the rule executes. The log.info() function takes a string parameter; complex strings by appending variables and literals together using the '.' operator can be constructed (Figure 25).

The functions `log.warn()` and `log.error()` are similar to `log.info()`. They prefix error log messages with a higher priority—either "WARN" or "ERROR". The Event Log viewer can be used to filter out low-priority messages.

Be careful when printing out connection data verbatim, because the connection data may contain control characters or other non-printable characters. Encode data using either `'string.hexEncode()'` or `'string.escape()`.

Administrators should use `'string.hexEncode()'` if the data is binary, and `'string.escape()'` if the data contains readable text with a small number of non-printable characters.

```
# Get the user name
$user = http.getCookie( "UserName" );

# Don't cache any pages by default:
http.cache.disable();

if( $isvaluable ) {
   # Do cache these pages for better performance.
   # Each user gets a different version of the page, so we need to cache
   # the page indexed by the user name.
   http.cache.setkey( $user );
   http.cache.enable();
}
```

Figure 22: Cache a different version of Web page content for each user.

```
if( $isvaluable && slm.conforming( "timer" ) < 70 ) {
   log.info( "User ".$user." needs priority" );
}
```

Figure 23: Example an alert condition used to write a custom log file entry.

```
$ tail ZEUSHOME/zxtm/log/errors
[20/Jan/2004:10:24:46 +0000]
INFO:VSName:Rule=RuleName:User Jack needs priority
```

Figure 24: Example entry in the Event Log.

```
$msg = "Received ".connection.getDataLen()." bytes.";
log.info( $msg );
```

Figure 25: Creating more complex custom log entries.

Conclusion

Pulse Secure Virtual Traffic Manager is a powerful toolkit for network and application administrators. This white paper described a number of techniques to use tools in the kit to solve a range of traffic valuation and prioritization tasks.

About Pulse Secure

Pulse Secure networking solutions help organizations achieve their critical business initiatives as they transition to a world where applications and information reside anywhere. Today, Pulse Secure is extending its proven data center expertise across the entire network with open, virtual, and efficient solutions built for consolidation, virtualization, and cloud computing. Learn more at www.pulsesecure.net.

Corporate and Sales Headquarters
Pulse Secure LLC
2700 Zanker Rd. Suite 200
San Jose, CA 95134
www.pulsesecure.net

Pulse Secure®