



Red Condor Verifier XML Syntax

0830-0023, Rev. 1

October 2009



Contents

Document Revisions	7
Verifier Syntax	9
VrfyXML Elements	11
List of Elements and Attributes.....	12
Connect Elements.....	12
Event Elements.....	13
LDAP Verifier Elements	14
SMTP Verifier Elements.....	15
Database Verifier Elements	16
Sub-Verifier Elements	17
The Vrfy Element	17
Generic/Shared Elements	18
Meta Elements	18
Connect Elements	19
element: <Timeout>	19
element: <Credentials>.....	20
element: <BackendMax>	21
element: <BackendIdleTime>	22
element: <Host>	23
element: <HostListOrder>	25
Connect Elements Example: LDAP	25
Connect Elements Example: SMTP VRFY	26
Connect Elements Example: DataBase	26
Connect Elements Example: Communigate	27
Connect Elements Example: Multi	27
Event Elements.....	28
element: <InterEnumTime>	28
element: <InterDiscTime>	30

Verifier Types	31
LDAP Verifiers	31
About LDAP Searches.....	32
element: <BaseDN>.....	33
element: <Filter>.....	33
element: <Canonical>.....	34
element: <StripPfx>.....	36
element: <EnumFilter>.....	36
element: <EnumTest>.....	37
element: <EnumTo>.....	38
element: <EnumFrom>.....	39
element: <DiscFilter>.....	39
element: <DiscAttr>.....	39
LDAP Example 1.....	40
LDAP Example 2.....	40
LDAP Example 3.....	41
SMTP Verifiers	41
element: <Threshold>.....	41
element: <Errmap>.....	42
element: <UseBrackets>.....	42
SMTP Examples.....	42
Communigate Verifier	43
Communigate Example.....	44
Database Verifiers	44
Query Placeholders.....	44
element: <Vendor>.....	45
element: <DBName>.....	45
element: <VrfyQuery>.....	45
element: <AuthQuery>.....	46
element: <DiscQuery>.....	47
element: <EnumQuery>.....	47
DataBase Examples.....	48
Multi Verifier	49
Sub-Verifier Attributes.....	50
Element <Domain>.....	51
Multi Verifier Example 1.....	51
Multi Verifier Example 2.....	52
Multi Verifier Example 3.....	53
Multi Verifier Example 4.....	53

Appendix A: Verifier Feature Implementation Status57

Appendix B: Configuration Element Summary59



Document Revisions

The table below shows revision changes of this document:

Revision	Date	Changes
03	09/04/09	<ul style="list-style-type: none">Added support for POP3 verifiers. See <i>VrfyXML Elements</i> (on page 11), <i>Connect Elements</i> (on page 12), <i>element: <Timeout></i> (on page 19), <i>element: <Host></i> (on page 23), <i>element: <InterEnumTime></i> (on page 28), <i>Verifier Types</i> (on page 31), <i>SMTP Verifiers</i> (on page 41), <i>SMTP Examples</i> (on page 42), <i>Appendix A: Verifier Feature Implementation Status</i> (on page 57), and <i>Appendix B: Configuration Element Summary</i>. (see "Appendix B: Configuration Element Summary" on page 59)



Verifier Syntax

Verifiers define a method for validating an email address and/or authenticating a user. Verifiers are used in domain configuration. They consist of settings used for communicating with the verification server.

The verifier service contains components that improve verification performance including:

- **Enumeration:** All valid addresses are retrieved from the verification server and cached locally to speed up verification and reduce the load on network resources.
- **Domain discovery:** The domains serviced by each verification server are determined and put in a list.

Verifiers are defined through the Red Condor Administrator Dashboard, or the Red Condor Provisioning API using the Red Condor Verifier XML Syntax (VrfyXML). The Provisioning API exposes additional settings not available in the Administrator Dashboard. Each verifier instance is described by a single `<vrfy>` element.



Note: XML is case sensitive. In general, XML element names in VrfyXML are CamelHumps case while attribute names are usually all lower case. CamelHumps case are also acceptable for attribute values. Also note that the XML comment `<!-- ... -->` is valid within VrfyXML.



Chapter 2

VrfyXML Elements

The **<Vrfy>** element is the top level XML element (document root) for each verifier. Inside of each **<Vrfy>** element is one verifier type element. Red Condor supports the following verifier types:

- LDAP
- RcptTo
- Postfix (SMTP VRFY)
- Communicate
- DataBase
- POP3
- Multi

The Multi verifier type is special. It can contain one or more of the other verifier types. When they appear inside of a **<Multi>** element, they are called sub-verifiers. The options available in configuring sub-verifiers are different from top-level verifiers. Also, a Multi verifier cannot contain another Multi verifier.

All other configurable data exists inside of the typed-verifier. Some configuration options are available in more than one type of verifier. Other configuration options are only available for a specific verifier type.

This document discusses the attributes of the **<Vrfy>** element first, then the shared elements available to more than one of the verifier types. Then, each verifier type has its own section describing which of the shared elements and which unique elements it may/must contain. Finally, there are some examples.

Appendix A: Verifier Feature Implementation Status (on page 57) shows the feature implementation status for the verifier types. *Appendix B: Configuration Element Summary* (on page 59) summarizes the elements that can appear in each type of verifier and how many may appear.

List of Elements and Attributes

The following tables list all elements and their attributes for each type of verifier, and gives a description and displays their valid options.

Connect Elements

Element	Attributes (A)/ Sub-Elements (S)	Description	Valid Options
<i>BackendIdleTime</i> (see "element: <BackendIdleTime>" on page 22)		Defines the number of seconds a connection can remain idle before it is automatically closed	Positive integer
<i>BackendMax</i> (see "element: <BackendMax>" on page 21)		The maximum number of connections allowed to the verifier server	Positive integer
	enumMax (A)	Limits how many of the available <BackendMax> connections can be used for address enumeration	A positive integer no greater than <BackendMax>, or an integer percentage
<i>Credentials</i> (see "element: <Credentials>" on page 20)		User name and password for access to the verification server	Maximum one user name/password set
	Username (S)	Name of the user	ASCII text
	Password (S)	Password of the user	Clear text
	EncPass (S)	Encoded password of the user	Password in the encryption format used by the verification server
<i>Host</i> (see "element: <Host>" on page 23)		Host name of the verification server	Host name or IP address
	secure (A)	The connection to the verification server is made using SSL	<p>true: Uses SSL encryption for secure connection.</p> <p>false: Does not use SSL encryption for secure connection.</p> <p>starttls: Use TLS for authentication requests. If TLS is not supported the request fails.</p> <p>trystarttls: Attempt to use Transport Layer Security (TLS) for authentication requests. If TLS is not supported, communication with the verification server is not encrypted.</p>
	allowInsecureAuth (A)	Whether or not the authentication request can be sent in clear text.	<p>true: Authentication requests can be sent unencrypted.</p> <p>false: Unencrypted authentication requests fail.</p> <p>Default value is true.</p>

	defaultRouteMatch (A)	Used to remove a specific <Host> element from consideration based on the default route of the appliance.	A regular expression that must match the default route of the appliance. If it does not match, the host is not considered at all when establishing connections to the servers defined in the <Host> elements.
	defaultRouteNoMatch (A)	Used to remove a specific <Host> element from consideration based on the default route of the appliance.	A regular expression that cannot match the default route of the appliance. If it does match, the host is not considered at all when establishing connections to the servers defined in the <Host> elements.
<i>HostListOrder</i> (see "element: <HostListOrder>" on page 25)		The algorithm used to select the next server to query	<p>Static: (Default) Selects the first host from the list. If it is unsuccessful, it selects the other hosts in order until one is successful.</p> <p>Rotate: Selects the next entry from the list of <Host> elements in a ring fashion.</p> <p>Shuffle: Randomly selects an entry from the list of <Host> elements.</p>
<i>Timeout</i> (see "element: <Timeout>" on page 19)		Maximum number of seconds between the time the application sends a request and it receives a response before the request is treated as a failure	<p>Positive integer. Defaults:</p> <p>LDAP: 10</p> <p>CommuniGate: 10</p> <p>DataBase: 10</p> <p>Postfix: 10</p> <p>RcptTo: 20</p> <p>POP3: 10</p>

Event Elements

Element	Attributes	Description	Valid Options
<i>InterDiscTime</i> (see "element: <InterDiscTime>" on page 30)		The execution frequency of the domain discovery query	In minutes. Non-negative integer. Default value is zero (no domain discovery). For Multi verifiers only.
<i>InterEnumTime</i> (see "element: <InterEnumTime>" on page 28)		The amount of time between address enumerations	In minutes. Non-negative integer. If set to zero, then address enumerations will not occur

LDAP Verifier Elements

Element	Attributes (A)/ Sub-Elements	Description	Valid Options
	defaults (A)	The set of sub-element default values for a particular LDAP server type	Generic Active Directory GroupWise Zimbra Domino
<i>BaseDN</i> (see "element: <BaseDN>" on page 33)		Specifies the Base Distinguished Name to use when searching for a email address and enumerating addresses	LDAP distinguished name
<i>Canonical</i> (see "element: <Canonical>" on page 34)		Specifies templates used by the application to determine how to construct the email address from an LDAP search result	%{attribute name} %{rciVRFYdomain} combined with string constants
<i>DiscAttr</i> (see "element: <DiscAttr>" on page 39)		Attributes to be returned from a domain discovery LDAP search	LDAP attribute name
<i>DiscFilter</i> (see "element: <DiscFilter>" on page 39)		LDAP filter, for sub-verifier only, that identifies the domains served by the LDAP server	LDAP query syntax
<i>EnumFilter</i> (see "element: <EnumFilter>" on page 36)		Specifies the LDAP search filter used for address enumeration	LDAP query syntax
<i>EnumFrom</i> (see "element: <EnumFrom>" on page 39)		For address enumeration, the attribute containing the alias address	LDAP attribute name
<i>EnumTest</i> (see "element: <EnumTest>" on page 37)		Used to constrain address enumeration search results returned by <EnumFilter>	
	Attribute (S)	Names an attribute to check	
	Test (S)	The condition required to keep a record	
	operator= (A)	The operator for the test condition	>,!=,ge,<,==,le,>=,gt,eq,<=,lt,ne
<i>EnumTo</i> (see "element: <EnumTo>" on page 38)		Specifies templates used by the application to determine how to construct the email address from an LDAP search result for address enumeration	%{attribute name} %{rciVRFYdomain} combined with string constants

<i>Filter</i> (see "element: <Filter>" on page 33)		The LDAP search filter to use for email address verification	Accepts the following placeholder variables: %D - The first domain component. %u - The username of the email address being verified. %d - The domain name of the email address being verified. %e - The equivalent of %u@%d .
<i>StripPfx</i> (see "element: <StripPfx>" on page 36)		Used to strip a prefix from an attribute value before constructing the email address to be returned	
	Attribute (A)	Target attribute	LDAP attribute
	Prefix (A)	String to strip off the attribute value	String

SMTP Verifier Elements

Element	Sub-Elements	Description	Valid Options
<i>Errmap</i> (see "element: <Errmap>" on page 42)		Maps SMTP response code to new value, based on the matching of a free-form regular expression	Must contain one <Text> and one <Code> element.
	Text	A Perl-compatible regular expression to be matched against each response line from the SMTP server	
	Code	The replacement response code for when the text is matched	Must contain a three digit integer
<i>Threshold</i> (see "element: <Threshold>" on page 41)		Minimum SMTP response code number that is considered a failure	Positive integer. Default is 300
<i>UseBrackets</i> (see "element: <UseBrackets>" on page 42)		Whether or not the email address is surrounded by angle brackets	true or false

Database Verifier Elements

Element	Attributes	Description	Valid Options
<i>AuthQuery</i> (see "element: <AuthQuery>" on page 46)		The SQL query issued for login authentication	SQL Placeholders %?U, %?D, and/or %?E, as well as either %?C or %?M. The query must returns one or zero rows. When zero rows are returned, the authentication fails.
<i>DBName</i> (see "element: <DBName>" on page 45)		The name of the database to connect to	String. Required
<i>DiscQuery</i> (see "element: <DiscQuery>" on page 47)		The SQL query issued to find all unique domains that have email addresses on the server	Returns one row per domain, each row having a single column containing the domain name
<i>EnumQuery</i> (see "element: <EnumQuery>" on page 47)		The SQL query to find all email addresses the database server will verify	The response should be one row for each email address and one for each alias
<i>Vendor</i> (see "element: <Vendor>" on page 45)		Required element that specifies the type of database	MySQL Postgres SQLite (reserved for testing purposes)
<i>VrfyQuery</i> (see "element: <VrfyQuery>" on page 45)		The SQL query used for email address verification	The first column is required and holds the canonical form of the requested email address. The second column is optional and returns a unique ID to be used when two-step authentication is performed.

Sub-Verifier Elements

Element	Attributes	Description	Valid Options
	optional	Whether or not to ignore errors during lookup and proceed to the next sub-verifier.	true or false
	service	Defines how the verifier is used.	omit: The results of address verification are inverted. If the sub-verifier finds a match, it will not search further and will return a failure. verify: The sub-verifier is only used for verification, and is ignored when processing authentication requests. all: The sub-verifier is used for both verification and authentication requests. Default value is all
	enumerate	Whether or not to enable enumeration for the sub-verifier.	true or false
<i>Domain</i> (see "Element <Domain>" on page 51)		Limits use of the sub-verifier to the specified domain	Domain name. Valid only when used with a sub-verifier.

The Vrfy Element

<Vrfy> is the top-level element for defining verifiers. It has one sub-element and four attributes: version (required), uid, name, and disable.

attribute: version

Value: string

All verifiers require the **version** attribute. It specifies the version of the VrfyXML syntax the given document is written in. All future versions of the application will be backwardly compatible.

attribute: uid

Value: string

The **uid** is the unique identifier of the verifier and is required for updating verifiers. It is not required for adding a verifier.

attribute: name**Value:** string

The **name** is a user-friendly identity of the verifier. It is used to identify the verifier in the Administrator Dashboard, and is optional. If used, it need not be unique.

attribute: disable**Value:** true/false

Setting **disable** to "true" makes the verifier unavailable for use. The verifier will still be stored in the database but switched off.

sub-element: <MetaData>

The **<MetaData>** sub-element is read-only and used by the application to store information. Do not modify the content of the **<MetaData>** sub-element when updating a verifier. **<Vrfy>** may contain any number of **<MetaData>** sub-elements. The **<Vrfy>** element also contains exactly one verifier type element (such as **<LDAP>** or **<RcptTo>**).

The following example shows only the structure of the document-level **<Vrfy>** element along with its attributes and a single **<MetaData>** sub-element.

```
<Vrfy version = "101.4635"
      name    = "My Favorite Verifier"
      uid     = "64378E86-A459-11DD-1273-09173F13E4C5">
  <MetaData>
    <Editable>true</Editable>
  </MetaData>
</Vrfy>
```

Generic/Shared Elements

The following elements can appear in more than one verifier type. They are divided into groups defined by the element's purpose:

- **Meta:** elements that affect the verifier itself.
- **Connect:** elements that relate to the establishment of application level connections to the verification server.
- **Event:** elements that control the frequency of periodic events.

Meta Elements

Meta elements are for Red Condor internal use only.

Connect Elements

The Connect elements, **<Timeout>**, **<Credentials>**, **<BackendMax>**, **<BackendIdleTime>**, **<Host>**, and **<HostListOrder>** relate to the establishment and management of a pool of TCP connections between the verifier and the verification server. **<BackendMax>**, **<Host>**, and **<HostListOrder>** work together to control the number of simultaneous connections managed by the application, and also define how the host to connect to is selected from a set of verification servers.

element: **<Timeout>**

Value: positive integer (in seconds)

A single **<Timeout>** element may appear in any top-level or sub-level verifier. If you do not provide a **<Timeout>** element, then each verifier type uses its own default value.

Type	Default Timeout (seconds)
LDAP	10
Communigate	10
DataBase	10
Postfix	10
RcptTo	20
POP3	10
Multi	(see below)

A timeout begins when the application sends a request to the verification server. It stops when the server responds with a complete message. In addition, because the number of connections is limited, issuing a request to a busy server can result in an unlimited wait for a free connection to become available.

When a timeout does occur, it is treated as an error. This error always results in the application closing the particular connection on which the timeout occurred. However, the application may reissue the same request to a different server, or it may return a failure for the current operation.

For Multi verifiers, you can specify zero or one **<Timeout>** element in the **<Multi>** element itself, and also zero or one **<Timeout>** element in each sub-verifier. If the **<Multi>** element itself contains no timeout, then each sub-verifier has a default value that can be overridden with a locally scoped **<Timeout>** element. If the **<Multi>** element does have a **<Timeout>**, then that **<Timeout>** becomes the default for any sub-verifier that does not contain its own **<Timeout>** element.

In the following example, a request to the "My Multi Verifier" verifier results in the following queries (stopping at the first match or error):

- LDAP lookup on ldap-1.company.com with timeout of 3 seconds
- RcptTo lookup on mail.company.com with timeout of 4 seconds
- LDAP lookup on ldap-2.company.com with timeout of 5 seconds

```
<Vrfy version = "101.4635"
  name      = "My Multi Verifier"
  uid       = "64378E86-A459-11DD-1273-09173F13E4C5">
  <Multi>
    <Timeout>4</Timeout>
    <LDAP>
      <Host>ldap-1.company.com</Host>
      <Timeout>3</Timeout>
    </LDAP>
    <RcptTo>
      <Host>mail.company.com</Host>
    </RcptTo>
    <LDAP>
      <Host>ldap-2.company.com</Host>
      <Timeout>5</Timeout>
    </LDAP>
  </Multi>
</Vrfy>
```

element: <Credentials>

Add a <Credentials> element for verifier types that require a username and password combination for access. Only one such element is allowed. The following verifier types support the <Credentials> element (both as top-level or as sub-verifiers in a <Multi> element): <LDAP>, <Communicate>, and <DataBase>.

Credentials cannot be shared across sub-verifiers by putting credentials at the top of a <Multi> verifier. A <Credentials> element must contain both a <Username> and one of the supported types of password element. The <Password> element can be encrypted or clear text.

sub-element: <Username>

The <Username> element can contain any text. For an LDAP verifier, it is the Distinguished Name (DN) to bind with. For a DataBase or Communicate verifier, it is simply the user name.

sub-element: <Password>

The <Password> element contains the password in clear text.

sub-element: <EncPass>

The **<EncPass>** element contains the encrypted password.

When you submit a verifier through the Red Condor Provisioning API, you can enter the clear text of the password into a **<Password>** element. It will then be transformed into an encrypted version and stored internally within an **<EncPass>** element. If you know the **<EncPass>** text corresponding to the password, you can craft your Red Condor Provisioning API submission with an **<EncPass>** element instead of a **<Password>** element.

If you do not supply credentials to an LDAP server, an anonymous bind is attempted. Because SQL servers can be configured to trust certain hosts without a password, you can also omit credentials from a DataBase verifier. Currently, connections to the Communicate server require a **<Credentials>** element. This requirement may change in a future release.

element: <BackendMax>

Value: positive integer

The **<BackendMax>** element gives the maximum number of simultaneous connections allowed between the verifier and the verification server. Each verifier maintains a pool of connections to its verification server. Whenever a request to verify, authenticate, discover, or enumerate is received, the application attempts to re-use an existing connection. Once such a command completes, the connection is stored in a pool of available connections.

Sometimes there are no available established connections, either because none have yet been established, or all the established connections are in use. In this case, the application adds a new connection to the pool and assigns it to the requesting command.

In order to control the load a verifier places upon the verification servers, the application imposes a limit on the maximum number of connections it will make for a verifier. This limit is stored in the **<BackendMax>** element. Note that, even though multiple hosts may be specified within a given verifier, it is the total number of connections across all hosts (even if some have zero connections) that count towards this maximum. When there are **<BackendMax>** active connections, additional requests for email address verification are enqueued until a connection becomes free. However the application will impose a time limit of 60 seconds on each verification request.

attribute: enumMax

Value: positive integer or integer percentage

The **<BackendMax>** element also may contain an optional **enumMax** attribute. The **enumMax** attribute limits how many of the available **<BackendMax>** connections can be used for enumeration. The **enumMax** attribute is either a positive integer no greater than **<BackendMax>** or an integer percentage.

Because enumeration commands receive priority in the job queue, servers that support enumerations over many connections in parallel could starve out all other requests until the enumeration completes. Enumerations can take up to ten minutes to complete for large domains. The **enumMax** attribute exists to prevent such starvation. Note that once an enumeration is successful, no verification servers are needed to service the typical verification request, even while a subsequent enumeration command is underway.

The following table shows the default connections for each verifier type:

Type	Default BackendMax	Default enumMax
LDAP	50	1
Communigate	5	80%
DataBase	50	1
Postfix	50	1
RcptTo	50	1
POP3	50	0
Multi	(see below)	(see below)

For Multi verifiers, you can specify zero or one **<BackendMax>** element in the **<Multi>** element itself, and also zero or one **<BackendMax>** element in each sub-verifier. If there is no **<BackendMax>** in the Multi verifier itself, then each sub-verifier has its normal default value that can be overridden with a locally scoped **<BackendMax>** element.

If the Multi verifier does have a **<BackendMax>** element, then it becomes the default for any sub-verifier that does not contain its own **<BackendMax>** element. The value of the **enumMax** attribute uses the above defaults when unspecified and propagates down automatically when it is inherited within a Multi verifier. See *Communigate Example* (on page 44) and *Multi Verifier Example 4* (on page 53) for examples of this usage.

element: **<BackendIdleTime>**

Value: positive integer (in seconds)

This element defines the amount of time (in seconds) a connection can remain idle before the application will automatically close it. The default value for **<BackendIdleTime>** is 90 seconds for all verifier types that maintain open connections. A sub-verifier in a Multi verifier can inherit the value of **<BackendIdleTime>** from its **<Multi>** parent. Red Condor recommends setting the **<BackendIdleTime>** to less than the **<InterEnumTime>**.

Once an enumeration has successfully completed, most connections will become idle, because most verification requests will be handled from the cached email information gathered during the enumeration. If these idle connections are not closed, then when the next enumeration begins, it may choose stale connections resulting in a transient failure of the enumeration.

element: <Host>

Value: IP address or host name

The <Host> element identifies the verification server. If there is more than one <Host> element, each is expected to serve the same set of data. One reason to have multiple <Host> elements is for high-availability.

For example, you can specify an LDAP primary server as one <Host> and its mirror in another. Another application of multiple hosts is to spread the load across many verification servers. However, if you have a domain where different verification servers send different data, then you do not need multiple <Host> entries in your verifier. Use a <Multi> verifier instead.

At least one <Host> element is required by <LDAP>, <Postfix>, <RcptTo>, <DataBase>, <POP3> and <Communicate> verifiers. When used with a DataBase <Vendor> of SQLite (testing only), the value of any <Host> element is ignored. The <Host> element is not valid in the <Multi> parent.

Each <Host> element contains a host-name or IP address followed by an optional colon and TCP port number. If the port number is not given, a default port number is assumed. See the table below for default ports.

The attributes of the <Host> element offer connection controls. These attributes are **secure**, **allowInsecureAuth**, and **defaultRouteMatch/defaultRouteNoMatch**.

attribute: **secure**

Value: true/false, starttls, trystarttls

When the **secure** attribute is set to true, the connection between the verifier and the verification server is encrypted. The **starttls** uses Transport Layer Security (TLS) for authentication requests. If TLS is not supported the request fails. The **trystarttls** attribute attempts to use TLS for authentication requests. If TLS is not supported, communication with the verification server is not encrypted.

The following table summarizes default port numbers for various types of verifiers, depending on the value of the **secure** attribute.

Type	Default Clear Port secure = false	Default Secure Port secure = true
LDAP	389	636
Communicate ¹	106	106
MySQL DB ²	3306	3306
PostgreSQL DB ²	5432	5432
SQLite DB	N/A	N/A
Postfix/RcptTo	25	465
POP3	995	110

1 Note that Communicate verifiers cannot listen on both a secure and an insecure port. The Communicate administrator selects the port. The standard practice for a Secured Communicate server is to run it on port 106. On Mac OS X, Communicate CLI's default port is 8106. Therefore use **<hostname>:8106** when connecting to a Communicate verifier running OSX.

2 Note that the default port number for a **<DataBase>** verifier depends upon the verifier **<Vendor>** element.

attribute: allowInsecureAuth

Value: true/false

The **allowInsecureAuth** attribute controls the sending of authentication requests through clear text. When "true", authentication requests can be sent unencrypted. When "false", authentication requests sent unencrypted fail. The default value is true.

attribute: defaultRouteMatch

attribute: defaultRouteNoMatch

Value: regular expression

The **defaultRouteMatch** and **defaultRouteNoMatch** attributes are used to remove a specific **<Host>** element from consideration based on the default route of the appliance. This feature is used to allow a single verifier definition to be replicated to hosts both behind a customer firewall (such as an appliance) and outside the customer's firewall (such as a Vx appliance).

The **defaultRouteMatch** attribute contains a regular expression that must match the default route of the appliance. If it does not match, the host is not considered at all when establishing connections to the servers defined in the **<Host>** elements. Likewise, the **defaultRouteNoMatch** attribute exists to omit a host from consideration if its regular expression argument matches the default route.

<Host> Element Example

Consider the following verifier example:

```
<Vrfy version = "101.4635"
  name      = "My LDAP Verifier"
  uid       = "64378E86-A459-11DD-1273-09173F13E4C5">
  <LDAP>
    <Host defaultRouteMatch="12\.32\.1\.[0-9]+">10.111.1.12</Host>
    <Host defaultRouteNoMatch="12\.32\.1\.[0-9]+">112.11.64.244</Host>
  </LDAP>
</Vrfy>
```

Using this verifier, if the command **ip route | grep default**, returns:

```
default via 118.221.76.126 dev eth0
```

then connections will only be made to 112.11.62.244. However if the same command returns:

```
default via 12.32.1.1 dev eth0
```

then connections will only be made to 10.111.1.12.

element: <HostListOrder>

Value: rotate/shuffle

Each time a new connection to a verification server is required, the list of potential hosts (the set of <Host> elements) is subjected to one of the three <HostListOrder> algorithms:

- **Static:** (Default) Selects the first host from the list. If it is unsuccessful, it selects the other hosts in order until one is successful.
- **Rotate:** Selects the next entry from the list of <Host> elements in a ring fashion.
- **Shuffle:** Randomly selects one entry from the list of <Host> elements.

Connect Elements Example: LDAP

The following example shows an LDAP verifier establishing up to 25 simultaneous connections to the LDAP server on host 1.2.3.4. If an attempt to connect to that LDAP server fails, only then will a connection to 1.2.3.5 be attempted. Thereafter, when new connections are needed, 1.2.3.4 will still be tried first, but the established connection to 1.2.3.5 will remain in the pool for jobs to use.

If an attempt to establish a connection takes longer than five seconds, it is considered a failed connection attempt. Likewise, if an established connection does not respond for more than five seconds, that connection is shut down. When enumerating, the first available connection in the pool of cached connections is chosen, and that connection alone is used to perform the entire enumeration procedure:

```

<Vrfy version = "101.4635"
  name      = "My LDAP Verifier"
  uid       = "64378E86-A459-11DD-1273-09173F13E4C5">
  <LDAP>
    <BackendMax>25</BackendMax>
    <Host>1.2.3.4</Host>
    <Host>1.2.3.5</Host>
    <Timeout>5</Timeout>
  </LDAP>
</Vrfy>

```

Connect Elements Example: SMTP VRFY

The following example shows a Postfix SMTP VRFY verifier that establishes up to 10 simultaneous connections to the SMTP servers on hosts 5.6.7.8 and 5.6.7.9. Each time a new connection is required, the application alternates the host.

If the connection to the host chosen by the application fails (or does not respond within ten seconds), then the application tries the other host. Only if all hosts fail will the command that requires the connection fail.

```

<Vrfy version = "101.4635"
  name      = "My SMTP VRFY Verifier"
  uid       = "64378E86-A459-11DD-1273-09173F13E4C5">
  <Postfix>
    <Host>5.6.7.8</Host>
    <Host>5.6.7.9</Host>
    <HostListOrder>Rotate</HostListOrder>
  </Postfix>
</Vrfy>

```

Connect Elements Example: DataBase

The following example shows a DataBase verifier that establishes up to 50 simultaneous connections to the MySQL data base called email_addresses hosted on MySQL database servers listening at 10.11.12.1, 10.11.12.2 and 10.11.12.3. Each time a new connection is needed, a random one is selected from these three.

If a connection to the randomly selected host fails due to an error, then the application randomly tries one of the other two hosts. If that one fails as well, the third and final host is tried. Only if all three fail to connect properly will the pending command fail.

If two of the three hosts are unresponsive for some time, then the pool of connections consists of connections to the one responsive host. The application will not renegotiate connections to the failed hosts unless some of the cached connections for the functional host fail or are shutdown due to idleness. Enumerations take place entirely on the first free cached connection.

```

<Vrfy version = "101.4635"
  name      = "My MySQL Verifier"
  uid       = "64378E86-A459-11DD-1273-09173F13E4C5">
  <DataBase>
    <Vendor>mysql</Vendor>
    <DataBase>email_addresses</DataBase>
    <Host>10.11.12.1</Host>
    <Host>10.11.12.2</Host>
    <Host>10.11.12.3</Host>
    <HostListOrder>Shuffle</HostListOrder>
  </DataBase>
</Vrfy>

```

Connect Elements Example: Communicate

The following example shows a Communicate verifier that establishes up to 192 connections to host 8.8.8.71 on TCP ports 106 and 601. It will try port 106 first and only if that connection fails or takes longer than six seconds will it try the alternate port number 601.

If both ports fail or timeout, the pending command will fail. Once a connection is established, it is maintained. When enumerating email addresses, up to 144 connections are established and used by the enumeration process, leaving up to 48 connections available for verification commands.

```
<Vrfy version = "101.4635"
  name = "My Communicate Verifier"
  uid = "64378E86-A459-11DD-1273-09173F13E4C5" >
  <Communicate>
    <Timeout>6</Timeout>
    <BackendMax enumMax="75%">192</BackendMax>
    <Credentials>
      <Username>zed</Credentials>
      <Password>zed's dead</Password>
    </Credentials>
    <Host>8.8.8.71</Host>
    <Host>8.8.8.71:601</Host>
  </DataBase>
</Vrfy>
```

Connect Elements Example: Multi

The following example shows a Multi verifier that connects to many different LDAP directories. A copy of each directory is hosted on one or more LDAP servers using anonymous binding for each directory. When a request to verify an email address is received, the first available cached connection to the host 1.2.3.4 or 1.2.3.5 is queried with search base of dc=redcondor,dc=com using a one-second timeout.

If this query returns no hits, then the first cached connection to 1.2.3.6 is queried with search base of dc=redcondor,dc=net using an eight-second timeout. If this query returns no hits then the first cached connection to the host 1.2.3.7, 1.2.3.8 or 1.2.3.9 is searched. The search base is the LDAP server's published default search base. If this query returns no hits, the email address is not verified. If any one of the three searches times out or returns an error, then a "failure to verify" error is reported for that address.

When establishing connections, this verifier allows only one active connection to either host 1.2.3.4 or 1.2.3.5. If that connection times out or has an error, the other server is tried. If an error or timeout occurs during connection establishment to host 1.2.3.4 or 1.2.3.5, then the other server will be tried immediately. If the error occurs during connection use, the pending command fails.

This verifier also manages 18 separate connections to the LDAP server on host 1.2.3.6. Lastly, this verifier manages up to three simultaneous connections across the host set 1.2.3.7, 1.2.3.8 or 1.2.3.9. It will always try 1.2.3.7 first. If that connection attempt fails, it tries 1.2.3.8. Only if both 1.2.3.7 and 1.2.3.8 fail will it attempt a connection to host 1.2.3.9. All connection attempts will have eight-second timeouts.

When enumerating email addresses, one active connection from host 1.2.3.4 or 1.2.3.5 is used and, simultaneously, one active connection to 1.2.3.6 is used, and one active connection to any of host 1.2.3.7, 1.2.3.8, or 1.2.3.9 is used. The result from all three directory services are combined to yield the enumeration of this Multi verifier.

```
<Vrfy version = "101.4635"
  name = "My LDAP Verifier"
  uid = "64378E86-A459-11DD-1273-09173F13E4C5">
  <Multi>
    <BackendMax>3</BackendMax>
    <Timeout>8</Timeout>
    <LDAP>
      <Host>1.2.3.4</Host>
      <Host>1.2.3.5</Host>
      <BackendMax>1</BackendMax>
      <Timeout>1</Timeout>
      <HostListOrder>Rotate</HostListOrder>
      <BaseDN>dc=redcondor,dc=com</BaseDN>
    </LDAP>
    <LDAP>
      <Host>1.2.3.6</Host>
      <BackendMax>18</BackendMax>
      <BaseDN>dc=redcondor,dc=net</BaseDN>
    </LDAP>
    <LDAP>
      <Host>1.2.3.7</Host>
      <Host>1.2.3.8</Host>
      <Host>1.2.3.9</Host>
    </LDAP>
  </Multi>
</Vrfy>
```

Event Elements

Verifiers can have two repeating tasks that you can control using the events elements. These events are periodic email address enumeration and periodic domain discovery. The frequency of their occurrence is controlled by the **<InterEnumTime>** and **<InterDiscTime>** elements, respectively.

The **<InterEnumTime>** and **<InterDiscTime>** represent the average length of time between enumerations or discoveries. For example, if the **<InterEnumTime>** is set to 60 seconds, then the moment the enumeration completes a new enumeration will begin between 30 and 90 seconds later. Regardless of how small you set the **<InterEnumTime>**, there are never two overlapping enumerations.

element: <InterEnumTime>

Value: positive integer

The **<InterEnumTime>** element controls the delay between enumerations. If set to zero, then address enumeration will not occur.

For verifiers with periodic enumeration, all email addresses are retrieved from one of the verification servers and stored in a cache of valid email addresses. This enumeration is performed simultaneously from many servers in the case of a Communigate verifier, or from each sub-verifier in a Multi verifier . This cache allows the application to respond instantly to verification requests.

Although this address enumeration places considerable load on the verification servers, it can shield servers from the severe load of a Directory Harvest Attacks (DHA). You can expect enumeration of large LDAP servers to take up to ten minutes. If the enumeration procedure encounters an error, the entire database of enumerated email addresses is immediately discarded.

The following table summarizes the default `<InterEnumTime>` values for each verifier type. Note that when defining a Multi verifier, the `<InterEnumTime>` element is not allowed within a sub-verifier, and the default `<InterEnumTime>` is zero (disabled).

To enumerate a Multi verifier, you must explicitly request automatic enumeration by placing an `<InterEnumTime>` element as a child of the `<Multi>` parent element. Enumeration of all sub-verifiers of a Multi verifier is initiated in parallel.

Type	Default <code><InterEnumTime></code>
Communigate	60
DataBase	30
LDAP	20
Multi	0 (disabled) with inheritance
RcptTo	N/A
Postfix	N/A
POP3	N/A

element: <InterDiscTime>

Value: positive integer

Periodic domain discovery is only used in a Multi verifier. For verifiers with periodic domain discovery, each sub-verifier contacts each of its servers and generates a list of the domains that each server services. Domain discovery is used to limit the sub-verifiers used when verifying email addresses.

Before a domain discovery has been successfully completed, all requests to verify email addresses within a Multi verifier are sent to every verification server. Once a domain discovery is complete, a request to verify an email address is only sent to those sub-verifiers that service the domain of that email address.

The default value of <InterDiscTime> is zero (no domain discovery).

Chapter 3

Verifier Types

The verifier type sub-element within the **<Vrfy>** element defines the type of verifier. The following sections describe the broad classes of verifier, how they function, and the specific XML elements used to configure them. Verifier types include:

- LDAP
- RcptTo
- Postfix (SMTP VRFY)
- Communicate
- DataBase
- POP3
- Multi

LDAP Verifiers

An **<LDAP>** sub-element defines connections to LDAP servers such as Microsoft Active Directory, Zimbra's built-in LDAP server, the GroupWise built-in LDAP service, Lotus Domino, and others. An **<LDAP>** sub-element requires at least one **<Host>** element.

The LDAP verifier configuration contains the following optional elements:

- **<BaseDN>**
- **<Filter>**
- **<Canonical>**
- **<EnumFilter>**
- **<EnumFrom>**
- **<EnumTo>**
- **<EnumTest>**
- **<DiscFilter>**

- `<DiscAttr>`
- `<StripPfx>`

Attribute: defaults

The LDAP verifier type contains several sets of default values for the above sub-elements. You can select a specific set of defaults by including the attribute **defaults** in the `<LDAP>` element itself. There are currently five sets of predefined values corresponding to five different LDAP server types:

- Generic
- Active Directory
- GroupWise
- Zimbra
- Domino

If you do not specify the attribute **defaults**, then Generic is assumed. Regardless of the LDAP type, when you include sub-element, the default value for that element is overridden.

About LDAP Searches

When verifying an email address, an `<LDAP>` verifier will issue an LDAP search command. An LDAP search requires a BaseDN, a scope, and a filter. You can control which BaseDNs are searched by providing a set of `<BaseDN>` elements.

If you do not provide the `<BaseDN>` elements, then the application will try to determine a search base to use by querying the LDAP server Root DSE. The sub scope is always used.

The filter is defined in the `<filter>` element or the default filter is used if there is none. You can also specify an `<EnumFilter>` element containing a search filter to use when enumerating email addresses.

The verification command is supposed to find the canonical form of the requested address (revealing aliases). But which of the multitude of attributes within an LDAP search result record contains the primary SMTP address and which are the aliases?

There is no globally accepted standard, so you can solve the problem by configuring a list of format strings that reference the LDAP record attributes to construct the canonical form. LDAP verifiers accept a list of `<Canonical>` elements, each of which specifies one possible canonical form for the email address.

When the record from a search result is received, each canonical-form template is populated with information from that record. If any blanks cannot be filled, that canonical form did not match. If all the given canonical forms fail to match, the whole record is considered a search miss and the next record from the LDAP search is examined. The first record that matches one of the defined canonical forms by binding every variable with attributes from the record is the one returned to the application.

element: <BaseDN>

Value: distinguished name

The <BaseDN> element specifies the base distinguished name to use when searching for valid user email addresses and enumerating addresses. You may have unlimited <BaseDN> elements.

Any attempt to verify a user email address results in a separate LDAP search for each BaseDN until the user is found. Therefore, placing the most common <BaseDN> at the top of the list can gain a performance boost. You can supply a blank <BaseDN> to some LDAP servers, such as GroupWise. If you provide no <BaseDN> element, then the application will try the following attributes from RootDSE of the server:

- **DefaultNamingContext**
- one of the **NamingContexts**
- the server's name (dsaName)

To be eligible, a **NamingContexts** value must be the shortest non-empty **NamingContexts** value that has either a **dc** or **o** component.

element: <Filter>

Value: LDAP query

The <LDAP> verifier supports one <Filter> element. This element specifies the LDAP search filter to use to verify email addresses. The search is repeated for every BaseDN (in order). The results of this search are a set of records which are then matched to the <Canonical> forms.



Note: The LDAP filter Syntax is case insensitive. An asterisk (star character (*)) denotes any span of characters.

The <Filter> element accepts the following placeholder variables:

- **%D** - The first domain component. For example, if you are verifying "adamsmith@internal.redcondor.com", then %D is "internal".
- **%u** - The username of the email address being verified. For example, if you are verifying "adamsmith@redcondor.com", then %u is "adamsmith".
- **%d** - The domain name of the email address being verified. For example, if you are verifying "adamsmith@redcondor.com", then %d is "redcondor.com".
- **%e** - The equivalent of %u@%d . This greatly improves the filter speed, especially during enumeration.

The default values for each type of LDAP server are:

Type	Default LDAP Search Filter
Generic	(mail=%e)
GroupWise	(cn=%u)
ActiveDirectory	(&((proxyAddresses=smtp:%e)(userprincipalname=%e)(mail=%e))(!(msExchUserAccountControl=2)))
Zimbra	(&(zimbraMailStatus=enabled)((mail=%e)(zimbraMailAlias=%e)))
Domino	(&((objectClass=dominoGroup)(objectClass=dominoServerMailInDatabase)(objectClass=person))((mail=%e)(&(mail=*@%d)(uid=%u))(uid=%e)))

element: <Canonical>

The <Canonical> element is used when an LDAP directory has inconsistent attributes across all records. For example, some records have a mail attribute, while others have a **proxyAddresses** attribute.

The <Canonical> element specifies a template used by the application to construct the email address from an LDAP search result. You can have multiple <Canonical> elements. Each LDAP search can return a set of records. Each record contains many attributes, each with a set of values.

The syntax for the <Canonical> element contains the literal text of the canonical form of the email address, as well as the special sequence `%{attribute_name}` that references an attribute in the LDAP record. Each variable reference is replaced with the value of the LDAP record attribute with that name. The special sequence `%{rciVRFYdomain}` is replaced with the domain from the original verification query.

For a <Canonical> element to match a record, all the attribute references must be bound to a value from the record. If a <Canonical> element contains a reference to an attribute that does not exist in the record, that record cannot match that <Canonical> element. When examining each record the search returns, each <Canonical> element is tried in order. The first match is the email address returned.

The default values for each type of LDAP server are:

Type	Default LDAP Canonical Form(s)
Generic	%{mail}
GroupWise	%{mail}
ActiveDirectory	%{mail} %{proxyAddresses}
Zimbra	%{zimbraMailDeliveryAddress} %{mail} %{zimbraMailAlias}
Domino	%{mail}

If you try to verify `username@domain.com` with a Zimbra LDAP verifier, the application performs an LDAP search against each BaseDN with a search filter that finds all records that have any **zimbraMailStatus** attributes equal to `enabled`, and that have either:

- a **mail** attribute with the value `username@domain.com`.
- or*
- a **zimbraMailAlias** attribute with the value `username@domain.com`.

Those records will each go through the following test:

1. If there is a **zimbraMailDeliveryAddress** attribute, then return the value of the first such attribute as the canonical form.
2. If there is a **mail** attribute, then return the value of the first such attribute as the canonical form.
3. If there is a **zimbraMailAlias** attribute, then return the value of the first such attribute as the canonical form.
4. Otherwise, check the next record returned from the search.

You can put literal text in a **<Canonical>** element. For example, you could define an LDAP verifier as:

```
<Vrfy version="101.4635">
  <LDAP>
    <Host>1.2.3.4</Host>
    <Filter>(mail=%u@%d)</Filter>
    <Canonical>{%uid}@my.mailserver.com</Canonical>
    <Canonical>{%mail}</Canonical>
  </LDAP>
</Vrfy>
```

The above verifier retrieves every record where the **mail** attribute is `username@domain.com`. Then the first record that contains a **uid** attribute, the attribute is concatenated with the text `@my.mailserver.com` and the result is returned.

In effect, the **mail** attribute is the alias for `username@my.mailserver.com` whenever there is a **uid** attribute. If there is no **uid** attribute, then the **mail** attribute is the canonical form for that mail address.

element: <StripPfx>

The <StripPfx> element strips off the prefix of an attribute before it is used in the construction of an email address. It controls the <Canonical> variable binding step. Each <StripPfx> element contains one <Attribute> and one <Prefix> sub-element.

sub-element: Attribute

The <Attribute> sub-element names an LDAP attribute to transform.

sub-element: Prefix

The <Prefix> element contains the prefix to strip off.

When trying to bind a <Canonical> variable that contains a reference to <Attribute>, if the value of that <Attribute> begins with the <Prefix> text, that text is stripped before substitution is done in the <Canonical> element. If the <Attribute> value does not match the <Prefix>, then that attribute value cannot be used to replace the reference in the <Canonical> element.

ActiveDirectory is the only server type with a default for <StripPfx>:

```
<StripPfx>
  <Attribute>proxyAddresses</Attribute>
  <Prefix>smtp:</Prefix>
</StripPfx>
```

The special checks for attribute prefixes defined by <StripPfx> elements are also applied when performing enumerations.

element: <EnumFilter>

The <EnumFilter> element specifies the LDAP search filter used for address enumeration. The search is repeated for each <BaseDN> element. If you do not specify the <EnumFilter> element, then it is calculated from the <Filter> element as follows:

The enumeration search filter will be set to the contents of the <Filter> element, except that instances of:

- %u@%d
- %u
- %d

are replaced by %A (in order).

The **%A** placeholder is initially set to *****, resulting in the selection of any record for which the requested attribute exists.

The following table shows the default **<EnumFilter>** elements for Zimbra, GroupWise, and Active Directory:

Server Type	Default
Zimbra	(&(zimbraMailStatus=enabled)((mail=%A)(zimbraMailAlias=%A)))
GroupWise	(objectclass=%A)
Active Directory	((proxyAddresses=smtp:%A)(mail=%A))

Some LDAP servers have a limit on the number of records returned by any search. If the server returns a special error code indicating too many search results, then the application runs the search repeatedly, each time replacing, **%A** in order, with the following (separated in this list by semicolons (;)):

a*; b*; c* ... z*; 0*; 1* ... 9*; _*; .*; -*

This substitution results in up to 39 separate searches. If any of these searches return the "too many results" error code, then that particular search is divided 39 ways again. For example if a search of (mail=s*) returns too many results, it would be retried as another 39 separate searches (mail=sa*) , (mail=sb*) , and so on.

The set of splits for the **%A** placeholder is retained from one enumeration run to the next. The split will not be re-merged unless the application is restarted or the verifier in question is altered.

Some LDAP servers, such as GroupWise and ActiveDirectory, perform very poorly when searched for attributes that do not have indices. Instead, use a broad filter as shown in the table above.

element: **<EnumTest>**

In the case where a broad filter is used for enumeration, the **<EnumTest>** element is used as a second filter by the application to narrow the search results to valid email addresses. Each **<EnumTest>** element has **<Attribute>** and **<Test>** sub-elements.

sub-element: **<Attribute>**

The **<Attribute>** sub-element names an attribute to check.

sub-element: <Test>

The <Test> sub-element has an **operator=** attribute that must be one of the 12 operators listed in the table below:

>	!=	ge
<	==	le
>=	gt	eq
<=	lt	ne

The given operator is used to test the value of the named attribute against the literal value in the <Test> sub-element.

Example:

ActiveDirectory LDAP verifiers have the following default <EnumTest> element:

```
<EnumTest>
  <Attribute>msExchUserAccountControl</Attribute>
  <Test operator="!=">2</Test>
</EnumTest>
```

Each record returned by the <EnumFilter> search is checked to see if it contains an **msExchUserAccountControl** attribute with a value of 2. If so, the record is ignored.

element: <EnumTo>

When validating an email address, an LDAP verifier finds potential records using the <Filter> element. No matter which clause within the filter matched, each record returned is treated equally. Each record is matched against the canonical forms in order. The first canonical form that matches is returned. Therefore, unless the address to be validated happens to match the canonical form that the verifier returns, it is automatically an alias.

In contrast, the enumeration filter, defined in the <EnumFilter> element, produces every record that might contain email addresses. Any record could contain many email addresses, which leaves open the question "which one is the true canonical form and which are the aliases?" The <EnumFrom> and <EnumTo> elements contain this information.

If you do not specify <EnumFrom> and/or <EnumTo>, then their values are the same as the set of <Canonical> elements.

Whenever a record is returned from the `<EnumFilter>` search, two sets of `<Canonical>` forms are matched against the record: the `<EnumFrom>` and the `<EnumTo>` elements. The `<EnumTo>` matches in a manner similar to the `<Canonical>` element. The first set of attributes that can be bound to all the placeholders in the `<EnumTo>` element selects the email address for that record. The `<StripPfx>` element rules are also applied to this mapping. Each `<EnumTo>` element is tried in series until one of them can fill in all the blanks.

The `<EnumTo>` element differs from the `<Canonical>` element when it contains the `%{rciVRFYdomain}` placeholder. For verification, that placeholder is populated with the domain of the email address you are verifying. However, when enumerating, that placeholder is replaced with every domain that the verifier services.

element: `<EnumFrom>`

Unless you use `%{rciVRFYdomain}`, each record in the search result will produce a single canonical form email address from the set of `<EnumTo>` elements. This email address can have many aliases. The entire set of matches created for all the `<EnumFrom>` elements is taken as the set of all aliases in the record.

So, rather than taking the first `<EnumFrom>` element that can match all its attributes, it includes every match. This rule also applies to attributes. For example, if an `<EnumFrom>` element has an attribute placeholder and the record contains three different values, then `<EnumFrom>` will generate three aliases for that record. *LDAP Example 3* (on page 41) shows the `<EnumFrom>` element.

element: `<DiscFilter>`

Domain discovery is the process by which an LDAP sub-verifier within a Multi verifier identifies which domains the LDAP server services by issuing a special LDAP search. The `<DiscFilter>` element contains the LDAP filter for this search.

The search is repeated for each `<BaseDN>` element. The results are collected into a set of domains according to the `<DiscAttr>` elements. There is no default for the `<DiscFilter>` element because many LDAP servers do not contain specific records with this information.

element: `<DiscAttr>`

When performing domain discovery, an LDAP verifier performs an LDAP search using the `<DiscFilter>` search filter. The records returned by the search are examined for each LDAP attribute named in a `<DiscAttr>` element. The full text value of all attributes are added to the set of discovered domains (duplicates are suppressed).

The following example shows an LDAP verifier that inspects the **name** and **suffixes** attributes of records that have an **objectClass=domainname** attribute:

```
<Vrfy version="101.4635">
```

```

<Multi>
  <InterDiscTime>20</InterDiscTime>
  <LDAP>
    <DiscFilter>(objectclass=domainname)</DiscFilter>
    <DiscAttr>name</DiscAttr>
    <DiscAttr>suffixes</DiscAttr>
  </LDAP>
</Multi>
</Vrfy>

```

LDAP Example 1

The following example shows an LDAP verifier with a custom search filter and custom mail address in canonical form. For each email user, if you verify email_user@address.com and there is a **cn** (common name) attribute with the value email_user, then it will return email_user@mycorp.com.

```

<Vrfy name="mycorp" version="101.4635">
  <LDAP>
    <Credentials>
      <Username>cn=ldapquery,o=mycorp</Username>
      <EncPass>53616c4218a60056</EncPass>
    </Credentials>
    <Host secure="true">people.mycorp.com</Host>
    <BaseDN>o=mycorp</BaseDN>
    <Filter>(cn=%u)</Filter>
    <Canonical>%{cn}@mycorp.com</Canonical>
  </LDAP>
</Vrfy>

```

LDAP Example 2

The following example shows an LDAP verifier with a custom search filter, many canonical forms to try, a prefix stripper, and two LDAP server hosts to try. It is similar to an ActiveDirectory verifier that also has email addresses stored in the **userPrincipalName** attribute.

```

<Vrfy name="myschool" version="101.4635">
  <LDAP>
    <Credentials>
      <Username>redcondor@myschool.edu</Username>
      <EncPass>53616c74374218a60056</EncPass>
    </Credentials>
    <Host defaultRouteMatch="116\..*">116.88.224.99</Host>
    <Host defaultRouteNoMatch="116\..*">10.10.60.13</Host>
    <BaseDN>DC=myschool,DC=edu</BaseDN>
    <Canonical>%{userPrincipalName}</Canonical>
    <Canonical>%{mail}</Canonical>
    <Canonical>%{proxyAddresses}</Canonical>
  </LDAP>
  <Filter>(&amp;(|(proxyAddresses=smtp:%u@d)(userPrincipalName=%u@d)(mail=%u@d))
  )(! (msExchUserAccountControl=2))</Filter>
  <EnumFilter>(|(proxyAddresses=smtp:%u@d)(userPrincipalName=%u@d)(mail=%u@d))<
  /EnumFilter>
    <EnumTest>
      <Attribute>msExchUserAccountControl</Attribute>
      <Test operator="!=">2</Test>
    </EnumTest>
    <StripPfx>
      <Attribute>proxyAddresses</Attribute>
      <Prefix>smtp:</Prefix>
    </StripPfx>
  </LDAP>
</Vrfy>

```

LDAP Example 3

The following example shows an LDAP verifier where **<EnumFrom>** is used to define the email aliases for email address enumeration.

```
<Vrfy name="mycorp" uid="791DA3AC-7A84-407C-769C-EA43AC53BB75
version="101.4635">
  <LDAP>
    <InterEnumTime>60</InterEnumTime>
    <Host secure="true">people.mycorp.com</Host>
    <Credentials>
      <Username>uid=red_condor,ou=people,o=mycorp,o=com</Username>
      <EncPass>53616c746564d6d42b</EncPass>
    </Credentials>
    <BaseDN>ou=People,o=mycorp,o=com</BaseDN>
    <Canonical>{%uid}@mycorp.com</Canonical>

<Filter>(& (mycorpMailLocalAddress=%u@d) (!(mycorpMailRoutingAddress=:fail:))
)</Filter>
  <EnumFrom>{%mycorpMailLocalAddress}</EnumFrom>
</LDAP>
</Vrfy>
```

SMTP Verifiers

A **<RcptTo>**, **<Postfix>**, and **<POP3>** sub-element in the **<Vrfy>** element defines connections to SMTP servers, such as Postfix, Microsoft Exchange server, CommuniGate MTA, Zimbra MTA, Lotus Domino, and many others. A **<RcptTo>**, **<Postfix>**, or a **<POP3>** sub-element requires at least one **<Host>** element.

The primary difference between **<RcptTo>**, **<Postfix>**, and **<POP3>** style verifiers is the set of commands sent to the SMTP server:

- The **<RcptTo>** submits the RCPT TO command to the server and checks the response code.
- The **<Postfix>** style verifier sends the VRFY command and checks the response code.
- The **<POP3>** sends the APOP command and checks the response code.

element: **<Threshold>**

Value: positive integer

The **<Threshold>** element specifies the maximum positive response code. If an SMTP response code is received that is greater-than or equal to this number, then it is treated as a failure. Otherwise, it is verified.

element: <Errmap>

The <Errmap> element allows you to map SMTP responses to new response codes based on a free-form regular expression. This mapping is useful for some servers that return good (200-range) responses for partial failures. In particular, the rule helps when the SMTP server is willing to forward the mail for you. The default value is none.

Each <Errmap> element must contain one <Text> and one <Code> sub-element.

sub-element: <Text>

The <Text> element contains a Perl-compatible regular expression to be matched against each response line from the SMTP server.

sub-element: <Code>

The <Code> sub-element must contain a three digit integer.

If the regular expression from the <Text> element matches, then the SMTP response code is treated as if it had been the number in the <Code> sub-element. This value is then compared to the <Threshold> element of the verifier to determine if the response is a success or failure.

element: <UseBrackets>

Value: true/false

The <UseBrackets> element is only valid for <RcptTo> verifiers. The default value is false. If set to true, angle brackets are used to surround the email address. For example, the SMTP command:

```
rcpt to:email@domain
```

will instead be sent as:

```
rcpt to:<email@domain>
```

Some SMTP servers require brackets, while others do not.

SMTP Examples

The following is an example of a <RcptTo> verifier with a shortened timeout and <UseBrackets> enabled:

```
<Vrfy uid="791DA3AC-7A84-407C-769C-EA43AC53BB75" version="101.4635">  
  <RcptTo>  
    <Host>28.66.33.251</Host>  
    <Timeout>20</Timeout>  
    <UseBrackets>true</UseBrackets>  
  </RcptTo>  
</Vrfy>
```

The following example shows a **<Postfix>** verifier that randomly selects 1 of 11 mail servers to connect to, waits 20 seconds for the responses, and treats any response from the server that contains the words "but will relay" as if it were a code 551 response instead (it will not verify the requested address).

```
<Vrfy uid="A779050F-8F43-C556-E119-800789F83322" version="101.4635">
  <Postfix>
    <Errmap>
      <Code>551</Code>
      <Text>but\s+will\s+relay</Text>
    </Errmap>
    <HostListOrder>shuffle</HostListOrder>
    <Host>131.81.42.5</Host>
    <Host>131.81.42.6</Host>
    <Host>131.81.42.7</Host>
    <Host>131.81.42.8</Host>
    <Host>131.81.42.9</Host>
    <Host>131.81.42.10</Host>
    <Host>131.81.42.11</Host>
    <Host>131.81.42.12</Host>
    <Host>131.81.42.13</Host>
    <Host>131.81.42.14</Host>
    <Host>131.81.42.15</Host>
    <Timeout>20</Timeout>
  </Postfix>
</Vrfy>
```

The following example shows a **<POP3>** verifier that will attempt to use TLS during the communication with the verification server.

```
<Vrfy uid="A779050F-8F43-C556-E119-800789F83311" version="101.3807">
  <POP3>
    <Host secure="trystarttls">smtp.domain.net</Host>
    <Comment>Used for POP authentication.</Comment>
  </POP3>
</Vrfy>
```

Communigate Verifier

The **<Communigate>** sub-element in the **<Vrfy>** element defines connections to the Communigate Administrative Command Line Interface (CLI), which normally listens on TCP port 106. Using the Communigate verifier is usually the most efficient way to interact with a Communigate server. Communigate is currently the only verifier type to take advantage of parallel enumeration, in which a large number of verification servers share the work of enumerating the email address on the server.

It is important to note that many Communigate servers close TCP connections when too many connections are attempted. It is easy to overwhelm the Communigate server default connection limit (5). Coupled with a long **<Timeout>** setting, the result can appear as an application malfunction. For this reason, the default **<BackendMax>** element of Communigate verifiers is quite small.

Some Communigate installations can handle many hundreds of simultaneous connections. For such sites, enumeration can be completed in a reasonable amount of time by using parallel enumeration across a very large set of connections (100 to 200).

The **<Communicate>** sub-element requires the following elements:

- At least one **<Host>** element
- One **<Credentials>** element

Communicate Example

The following is an example of a Communicate verifier with a maximum of 9 connections to the one Communicate server at 38.16.99.4. Eight of those connections are allowed for enumeration use. The password is sent in encrypted format.

```
<Vrfy version = "101.4635"
  name   = "My Communicate Verifier"
  uid    = "5903BBD8-A30A-B384-B854-96B1DF65AB6A">
  <Communicate>
    <BackendMax enumMax="8">9</BackendMax>
    <Credentials>
      <Username>redcondor@mailhost.mycompany.net</Username>
      <EncPass>53616c746374218a60056</EncPass>
    </Credentials>
    <Host>38.16.99.4</Host>
  </Communicate>
</Vrfy>
```

Database Verifiers

The **<DataBase>** sub-element of the **<Vrfy>** element defines connections to SQL based database servers over TCP sockets. Red Condor supports both MySQL and PostgreSQL.

The **<DataBase>** sub-element requires the following elements:

- One **<Vendor>** element
- At least one **<Host>** element
- One **<VrfyQuery>** element
- One **<DBName>** element.

Timeouts for DataBase verification servers are not implemented.

For each type of command that the application will execute, a single SQL statement must be defined in one of the elements **<VrfyQuery>**, **<AuthQuery>**, **<DiscQuery>**, and **<EnumQuery>**. These **<DataBase>**-specific elements are described below. The queries accept standard placeholders where data from (or calculated from) the request will be substituted. Each of the query types also has an expected set of columns that should be returned.

Query Placeholders

When writing SQL statements for the four query elements, use placeholders anywhere the actual data from an application command will be inserted. Do not surround the placeholder in quotes for string values. Quoting will be automatic, using prepared statements.

Placeholder	Description
:%?U	Username portion of an email address for verification or authorization commands.
:%?D	Domain portion of an email address (after the @ sign) in verification or authorization commands.
:%?E	The full email address in verification or authorization commands (user@domain)
:%?C	Encrypted form of the password, calculated with a random salt each time an authorization command is received.
:%?M	md5sum of the password, calculated each time an auth command is received.

element: <Vendor>

The <Vendor> element specifies the type of database. This element is required. The currently supported database vendors are:

- MySQL
- Postgres
- SQLite (reserved for testing purposes)

element: <DBName>

The <DBName> element specifies the name of the database to connect to. It is required.

element: <VrfyQuery>

The <VrfyQuery> element is the SQL query issued for email address verification. You will likely need the placeholders %?U, %?D, and/or %?E. When an email address is not verified, the query should return no rows. When an email address is verified by the query, one row is returned.

The first column is required and holds the canonical form of the requested email address. This may be equal to the requested email address. The second column is optional and returns a unique ID to be used when two-step authentication is performed (two-step is not normally used with DataBase verifiers).

The following snippet shows a very simple <VrfyQuery> element that ignores aliases and unique IDs:

```
<VrfyQuery>
SELECT email FROM emails WHERE email=%?E
</VrfyQuery>
```

element: <AuthQuery>

The <AuthQuery> element is the SQL query issued for login authentication. You will need the placeholders %?U, %?D, and/or %?E, as well as either %?C or %?M. The query returns one or zero rows. When zero rows are returned, the authentication fails.

When one row is returned there are two potential columns. The first holds the UNIX-style encrypted password. The second is a Boolean that is true if the authentication has succeeded. If the Boolean is false, and there is a non-NULL value in the first column, the application compares the password from the authentication command to the encrypted one that came from the query. If they match, then the authentication succeeds.

It is important to construct your <AuthQuery> so that plaintext passwords are not passed over the wire to the database server. Most database connections are not encrypted and can easily expose credentials if you are careless with your SQL. For this reason, there is no placeholder for the plaintext password received in the authentication command.

The proper way to check a database of encrypted passwords is to simply return only the first column. The proper way to check a database of plaintext passwords is to send the encrypted password in your SQL and compare it to an SQL function call that encrypts the plaintext column on the database server. This method is slow, but secure. The correct way to query a database with both kinds of data is to do both and return both columns.

The following snippet shows a very simple <AuthQuery> element that returns the encrypted user password from a password table:

```
<AuthQuery>
  SELECT crypted
     FROM crypted_passwords
     WHERE mail_domain=?D AND username=?E
</AuthQuery>
```

The next snippet shows a method to write your <AuthQuery> element to check plaintext passwords on a MySQL server (other servers have different SQL functions available). Note that the first column (encrypted password) is always NULL and that the second column is the Boolean result of a comparison between the encrypted password (which is sent over the wire) and the encrypted version of the plaintext password (which is calculated on the server side). The calculation of the encrypted password on the SQL server uses the supplied encrypted password as a *salt*, which is standard practice (the prefix of the encrypted form holds the salt bits).

```
<AuthQuery>
  SELECT NULL, encrypt(cleartext_password,%?C)=?C
     FROM plain_passwords
     WHERE user_email=?E
</AuthQuery>
```

element: <DiscQuery>

The <DiscQuery> element is the SQL query issued to find all unique domains that are serviced on the verification server. This list of commands must include any domain (such as an alias) for which the server can ever successfully verify an email address. The query should return one row per domain, each row containing a single column containing the text of the domain name.

The following code snippet shows how to retrieve the unique list of domains from a table that has emails already separated into a user column and a domain column:

```
<DiscQuery>
SELECT DISTINCT domain FROM mail_table
</DiscQuery>
```

The next code snippet shows how to retrieve the unique list of domains from a table which has a column of full email addresses:

```
<DiscQuery>
SELECT DISTINCT mid(email_addr, instr(email_addr, '@')+1)
FROM email_table
</DiscQuery>
```

element: <EnumQuery>

The <EnumQuery> element is the SQL query to find all email addresses the database server will verify. The response should be one row for each email address and one for each alias. The columns of each row can contain:

- The email address or alias to be verified (the address which will be the argument of a verification command).
- The primary SMTP address. If this is non-NULL, then the first column contains an alias.
- The unique ID for that email address to be used in two-phase authentication. If NULL, two-phase authentication cannot be performed using enumerated data.
- The UNIX-style encrypted form for the user password. If NULL, authentication commands must use the <AuthQuery> element to check the password. If this is non-NULL, then authentication commands can be completed entirely within the application (after an enumeration has completed successfully). DataBase verifiers are the only type of verifier that can use internal authentication data to avoid querying the verification server. Do not create an <EnumQuery> element with SQL that will transport unencrypted passwords.

The following code snippet builds an enumeration from a table of email addresses and a table of aliases. There are no unique IDs nor are there passwords:

```

<EnumQuery>
SELECT email_aliases.from_email AS Lookup,
       email_accounts.email     AS Aliased_To
  FROM email_accounts INNER JOIN email_aliases
    ON email_accounts.email = email_aliases.to_email
UNION
  SELECT email,NULL
  FROM email_accounts;
</EnumQuery>

```

DataBase Examples

The following example shows a **<DataBase>** verifier that connects to the MySQL database named "customers" on TCP port 3306 (mydbserver.mycompany.net:3306) using username "redcondor" with a password. This verifier supports only the verification command.

```

<Vrfy uid="1874B47D-A461-8DB1-B897-D9B6805CD9CD" version="101.4635">
  <DataBase>
    <Vendor>mysql</Vendor>
    <DBName>customers</DBName>
    <Host>mydbserver.mycompany.net</Host>
    <Credentials>
      <Username>redcondor</Username>
      <EncPass>53616c7465645f4218a60056</EncPass>
    </Credentials>
    <VrfyQuery>
      SELECT A.email_address, A.row_id
      FROM email_table A, email_aliases B
      WHERE A.email_address=?E
      OR
      (B.alias=?E AND B.id=A.id)
    </VrfyQuery>
  </DataBase>
</Vrfy>

```

The following example shows a MySQL database verifier called "email_database". The database has two tables, "email_accounts" and "passwords". The "email_accounts" table has "id" and "address" columns. The passwords table has an "id" column which is a foreign key to the "id" column of the "email_accounts" table and a "password" column containing the unencrypted password for that user id.

Using enumeration, it constructs a result set containing encrypted passwords on-the-fly. There are no email aliases supported by this database, hence the NULL in the second column of the **<EnumQuery>** result set.

```

<Vrfy uid="C2EDEE82-B8B4-0FFE-93C6-B49ABB3D3B0B" version="101.4635">
  <DataBase>
    <Vendor>mysql</Vendor>
    <DBName>email_databae</DBName>
    <Host>mysql.mycompany.com</Host>
    <Credentials>
      <Username>redcondor</Username>
      <EncPass>53616c7465645f218a60056</EncPass>
    </Credentials>
    <VrfyQuery>
      SELECT address,id
      FROM email_accounts
      WHERE address=?E
    </VrfyQuery>
    <AuthQuery>
      SELECT NULL,encrypt(passwords.password,%?C)=?C
      FROM email_accounts LEFT JOIN passwords
      ON email_accounts.id=passwords.id
      WHERE email_accounts.address=?E
    </AuthQuery>
    <DiscQuery>
      SELECT DISTINCT mid(address, instr(address,'@')+1)
      FROM email_accounts
    </DiscQuery>
    <EnumQuery>
      SELECT email_accounts.address,
      NULL,
      email_accounts.id,
      encrypt(passwords.password, rpad(conv(rand()*32674,10,16), 2,'x'))
      FROM email_accounts LEFT JOIN passwords
      ON email_accounts.id = passwords.id
    </EnumQuery>
  </DataBase>
</Vrfy>

```

Multi Verifier

A **<Multi>** element is a sub-element of the **<Vrfy>** element. It contains any number of **<LDAP>**, **<RcptTo>**, **<Postfix>**, **<Communicate>**, and **<DataBase>** sub-verifiers. When a request to verify an email address is received by a Multi verifier, it queries each of its sub-verifiers in their defined order. Use a Multi verifier whenever you have one or more domains serviced by more than one server.

For example, a school might use a student-administered LDAP server to handle all the student email addresses. There is a faculty LDAP server to handle email addresses for the employees. They both require different credentials to bind to each server, and they each use a different list of BaseDNs to search. Yet, when a request to verify an email address comes in, it always contains "@school.edu". You need a single verifier to be mapped to the school.edu domain that searches each of the subordinate verifiers in turn. Multi verifiers do this.

To create a Multi verifier, wrap the sub-verifiers inside of **<Multi> ... </Multi>** tags in a single verifier. The **<InterDiscTime>** and **<InterEnumTime>** elements are only allowed as child elements of a Multi verifier, no other verifier type or sub-verifier can contain these elements.

The following elements can be used both as children of a sub-verifier and also as children of the **<Multi>** verifier itself: **<Timeout>**, **<BackendMax>**, and **<BackendIdleTime>**.

These elements are inheritable. This means that if a Multi verifier specifies a value for one of these elements, then all sub-verifiers that do not also specify a value (called overriding the inheritance) will inherit the value of the parent **<Multi>** element. In this way you can set, for example, a **<Timeout>** used by all the sub-verifiers in the **<Multi>** using only a single **<Timeout>** element as a child of the **<Multi>** element itself.

The behavior of the **<InterEnumTime>** element is different in Multi verifiers. Without this element, it is assumed that all the sub-verifiers have an **<InterEnumTime>** of 0 (zero). That is, enumeration is disabled by default on all the sub-verifiers. Because no sub-verifier is allowed to have the **<InterEnumTime>** element, the only way to get enumeration of sub-verifiers is to enumerate all sub-verifiers.

Sub-Verifier Attributes

The **optional** and **service** attributes apply to all sub-verifiers.

Attribute: optional

Value: true/false

When a verifier is used as a sub-verifier, you may add the **optional** attribute in the sub-verifier element (such as, **<LDAP>** or **<DataBase>**). The **optional** attribute is used to cause the Multi verifier to ignore errors during lookup and proceed to the next sub-verifier. Without this argument, a server error (such as lost connection to the LDAP server, or timeout to the SMTP server) results in a lookup failure because without the ability to query all the sub-verifiers, a negative verification response cannot be guaranteed correct.

On the other hand, in the school.edu example, if you set the student LDAP server as **optional="true"** and that LDAP server crashes, then all the student email addresses will appear to be invalid because the faculty LDAP server would report them as bad email addresses. Only mark sub-verifiers as **optional** if they are not needed to ensure the validity of a verification request.

Attribute: service

Value: omit/verify/all

Sub-verifiers can also have a **service** attribute. The default is all. When set to **service="omit"** the results of address verification are inverted. That is to say, if verifying **email@domain** and the sub-verifier finds a match, then the **Multi** verifier will not search subsequent sub-verifiers and will report the address as not-verified (550).

Conversely, when an email address is not matched by the **service="omit"** verifier, then the search continues with the next sub-verifier. Placing a sub-verifier with **service="omit"** as the last verifier will therefore have no effect. With the **service="verify"** argument, that particular sub-verifier is only used for verification, and is ignored when processing authentication requests. When the **service** attribute is set to all, then the sub-verifier is used for both verification and authentication requests.

Attribute: enumerate**Value: true/false**

The enumerate attribute determines whether or not to enable enumeration for the sub-verifier.

Element <Domain>

Sub-verifiers can have child **<Domain>** elements that force them to be used only when the domain being queried matches one of the **<Domain>** elements. For example, a Multi verifier mapped to domains a.com, b.com, and c.com.

If it has two sub-verifiers, you could assign domains a.com and b.com to be handled by the first sub-verifier and domains b.com and c.com to be handled by the second sub-verifier. Then when a request to verify an email address in a.com is received, only the first sub-verifier will be consulted.

When a request to verify an address in the b.com domain is received, then both sub-verifiers will be consulted (in order). Lastly, emails in the c.com domain will only search the second sub-verifier. See *Multi Verifier Example 2* (on page 52) for more information.

Multi Verifier Example 1

The following Multi verifier contains two LDAP sub-verifiers, each one with a primary and backup server, and a list of BaseDNs to search. Both use default queries for Active Directory servers.

In this Multi verifier, the search request will first be sent to host 216.141.147.111 (or 216.141.147.110 if 216.141.147.111 is unavailable). If a match is found, the result is that the email address is verified. No further search is performed.

If no match is found, then host 216.141.147.113 (or 216.141.147.112 if .113 is unavailable) is tried. Only if this query also fails will the address be reported as not verified. Both of the sub-verifiers will be enumerated, on-average, every 50 minutes.

Because this Multi verifier does not contain an **<InterDiscTime>** element, the Multi verifier is unaware of which domains are serviced by each sub-verifier and all searches will try the first sub-verifier. If there is no match, it searches the second sub-verifier. If any sub-verifier has an error, the verification query returns an error.

```

<Vrfy name="ben` `"
  uid="64378E3C-A459-11DD-1273-09173F13E4C5"
  version="101.4635">
  <Multi>
    <InterEnumTime>50</InterEnumTime>
    <LDAP defaults="ActiveDirectory">
      <Host>216.141.147.111</Host>
      <Host>216.141.147.110</Host>
      <Credentials>
        <Username>Red Condor</Username>
        <EncPass>53616c218a60056</EncPass>
      </Credentials>
      <BaseDN>OU=ADC Staging,DC=ben,DC=pri</BaseDN>
      <BaseDN>OU=Adjunct Faculty,DC=ben,DC=pri</BaseDN>
      <BaseDN>OU=BEN Faculty,DC=ben,DC=pri</BaseDN>
      <BaseDN>OU=BEN Staff,DC=ben,DC=pri</BaseDN>
      <BaseDN>OU=BEN Technical,DC=ben,DC=pri</BaseDN>
      <BaseDN>OU=Service Accounts,DC=ben,DC=pri</BaseDN>
      <BaseDN>OU=Student Workers,DC=ben,DC=pri</BaseDN>
      <BaseDN>CN=Users,DC=ben,DC=pri</BaseDN>
      <BaseDN>CN=Microsoft Exchange System Objects,DC=ben,DC=pri</BaseDN>
    </LDAP>
    <LDAP defaults="ActiveDirectory">
      <Host>216.141.147.113</Host>
      <Host>216.141.147.112</Host>
      <Credentials>
        <Username>Red Condor</Username>
        <EncPass>53616c7465644218a60056</EncPass>
      </Credentials>
      <BaseDN>OU=ActiveStudents,DC=stuben,DC=ben,DC=edu</BaseDN>
      <BaseDN>OU=NewStudents,DC=stuben,DC=ben,DC=edu</BaseDN>
      <BaseDN>CN=Microsoft Exchange System
Objects,DC=stuben,DC=ben,DC=edu</BaseDN>
    </LDAP>
  </Multi>
</Vrfy>

```

Multi Verifier Example 2

The following Multi verifier will search only ldap://1.2.3.4 when the query is **user@a.com**. It will search only ldap://1.2.3.5 when the query is **user@c.com**. It will search first ldap://1.2.3.4. Then, if no match is found, it will search ldap://1.2.3.5 when the query is **user@b.com**.

```

<Vrfy version="101.4635">
  <Multi>
    <InterEnumTime>50</InterEnumTime>
    <LDAP defaults="Generic">
      <Domain>a.com</Domain>
      <Domain>b.com</Domain>
      <Host>1.2.3.4</Host>
    </LDAP>
    <LDAP defaults="Generic">
      <Domain>b.com</Domain>
      <Domain>c.com</Domain>
      <Host>1.2.3.5</Host>
    </LDAP>
  </Multi>
</Vrfy>

```

Multi Verifier Example 3

In the following Multi verifier, there are again two LDAP directories to search. Neither one will be enumerated because the **<Multi>** parent is missing the critical **<InterEnumTime>** element. When this Multi verifier is loaded on a server with a default route matching 172.30.1.*, then for each verification request, it will query server 10.111.1.12 then 10.10.1.243 if there was no match found.

When the same Multi verifier is loaded on a server that does not have a default route in 172.30.1.*, then it will try to connect to 168.11.164.244. When no match is found for an email address, the search will be sent to 168.11.164.243.

```
<Vrfy version="101.4635">
  <Multi>
    <LDAP defaults="ActiveDirectory">
      <Host defaultRouteMatch="172\.30\.1\.[0-9]+">10.111.1.12</Host>
      <Host defaultRouteNoMatch="172\.30\.1\.[0-9]+">168.11.164.244</Host>
      <credentials>
        <Username>redcondor@qqab.local</Username>
        <EncPass>53616c74b374218a60056</EncPass>
      </credentials>
      <BaseDN>DC=sites,DC=QQAB,DC=local</BaseDN>
    </LDAP>
    <LDAP defaults="ActiveDirectory">
      <Host defaultRouteMatch="172\.30\.1\.[0-9]+">10.10.1.243</Host>
      <Host defaultRouteNoMatch="172\.30\.1\.[0-9]+">168.11.164.243</Host>
      <credentials>
        <Username>redcondor@qqab.local</Username>
        <EncPass>53616c7465645374218a60056</EncPass>
      </credentials>
      <BaseDN>DC=QQAB,DC=local</BaseDN>
    </LDAP>
  </Multi>
</Vrfy>
```

Multi Verifier Example 4

The following Multi verifier uses three separate sub-verifiers. One is a Communicate server, two are Active Directory servers. All three will be enumerated on-average, every forty minutes. Also, on average, every 20 minutes all three will be queried to learn the set of domains they service.

The Communicate server uses up to 210 simultaneous connections for general use, of which 200 can be devoted to an enumeration. The two LDAP servers will each never have more than 10 simultaneous connections owing to the inherited **<BackendMax>** child of the **<Multi>** element.

When verifying against this Multi verifier:

- The set of domains serviced by each of the sub-verifiers will be queried the moment the verifier is loaded from the database. In the LDAP verifiers, the domain discovery will be performed by finding every unique uPNSuffixes and **name** attribute for all records returned by searching with the filter (**objectclass=organizationalUnit**). The Communicate verifier uses default searches to find all the domains which it services.

- For each query, only those sub-verifiers that have reported that they service the queried domain will be searched, the others will be treated as though they were not present.
- When searching the Communicate sub-verifier:
 - If there is a verification server error (for example "TCP connection refused"), then an error is returned.
 - If the name is found, then the query succeeds.
 - If the name is not found:
 - If no other verifiers service the domain, then the query fails.
 - Otherwise, the next verifier that services the domain will be tried.
- When searching the first LDAP verifier:
 - If the name is found, then the query succeeds.
 - If the name is not found or if there is an error (because it is marked "optional"):
 - If the second LDAP sub-verifier services the domain, then it will be searched.
 - Otherwise, the query fails.
- When searching the last LDAP verifier:
 - If there is an error, then the query temporarily fails.
 - If the name is found, then the query succeeds.
 - Otherwise the query fails.

```

<Vrfy version="101.4635">
  <Multi>
    <InterDiscTime>20</InterDiscTime>
    <InterEnumTime>40</InterEnumTime>
    <BackendMax>10</BackendMax>
    <Communigate>
      <BackendMax enumMax="200">210</BackendMax>
      <Host>208.42.176.203</Host>
      <Credentials>
        <Username>redcondor</Username>
        <EncPass>53616c74656474218a60056</EncPass>
      </Credentials>
    </Communigate>
    <LDAP defaults="ActiveDirectory" optional="true">
      <Host secure="true">216.245.171.6</Host>
      <Credentials>
        <Username>cn=_exchange,ou=Customers,dc=agiliti,dc=net</Username>
        <EncPass>53616c7465645218a60056</EncPass>
      </Credentials>
      <BaseDN>ou=Customers,DC=agiliti,DC=net</BaseDN>
      <DiscFilter>(objectclass=organizationalUnit)</DiscFilter>
      <DiscAttr>uPNSuffixes</DiscAttr>
      <DiscAttr>name</DiscAttr>
    </LDAP>
    <LDAP defaults="ActiveDirectory">
      <Host secure="true">208.42.184.97</Host>
      <Credentials>
        <Username>CN=Red Condor,CN=Users,DC=vcollaborate,DC=com</Username>
        <EncPass>53616c746564b374218a60056</EncPass>
      </Credentials>
      <BaseDN>OU=Hosting,DC=vcollaborate,DC=com</BaseDN>
      <DiscFilter>(objectclass=organizationalUnit)</DiscFilter>
      <DiscAttr>uPNSuffixes</DiscAttr>
      <DiscAttr>name</DiscAttr>
    </LDAP>
  </Multi>
</Vrfy>

```



Appendix A

Appendix A: Verifier Feature Implementation Status

The following chart summarizes the feature implementation status for the various verifiers:

Type	Verification	Authentication	Enumeration	Discovery
LDAP	yes	yes	yes (parallel planned)	yes
RcptTo	yes	planned	no	no
Postfix	yes	planned	no	no
CommuniGate	yes	yes	yes (parallel)	yes
DataBase	yes	yes	yes	yes
POP3	no	yes	no	no
Multi	yes	yes	yes	yes

Appendix B

Appendix B: Configuration Element Summary

The following table summarizes which elements can be used (and how many are allowed) in each type of verifier. The table also indicates if the element is valid in sub-verifiers within a Multi verifier. The symbols in the table count the number of the given element type allowed in each of the possible locations. Here are their meanings:

Key	Description
0	This element is never allowed.
0-1	One of these elements is optionally allowed.
0+	Any number of that element is allowed, from none on up.
1	Exactly one of these elements is required.
1+	One of these elements is required, more are allowed.
SAME	The number of elements of the given type which are allowed in a sub-verifier is the same as the number allowed for a top-level verifier of the same type.
INHR	When in the <Multi>, then any sub-verifiers that do not have the given element will act as though they had the given element with the same value as the one in the <Multi> "parent" element.

Allowed XML Elements in VrfyXML (Part 1)

Type	Element	LDAP	Communigate	RcptTo	Postfix
Connect	Timeout	0-1	0-1	0-1	0-1
Connect	Credentials	0-1	1	0	0
Connect	BackendMax	0-1	0-1	0-1	0-1
Connect	BackendIdleTime	0-1	0-1	0-1	0-1
Connect	Host	1+	1+	1+	1+
Connect	HostListOrder	0-1	0-1	0-1	0-1
Multi	Domain	0	0	0	0
Event	InterEnumTime	0-1	0-1	0	0
Event	InterDiscTime	0	0	0	0
LDAP	BaseDN	0+	0	0	0
LDAP	Canonical	0+	0	0	0
LDAP	Filter	0-1	0	0	0
LDAP	StripPfx	0+	0	0	0
LDAP	EnumFrom	0+	0	0	0
LDAP	EnumTo	0+	0	0	0
LDAP	EnumFilter	0-1	0	0	0
LDAP	EnumTest	0+	0	0	0
LDAP	DiscFilter	0-1	0	0	0
LDAP	DiscAttr	0+	0	0	0
DB	Vendor	0	0	0	0
DB	DBName	0	0	0	0
DB	VrfyQuery	0	0	0	0
DB	AuthQuery	0	0	0	0
DB	EnumQuery	0	0	0	0
DB	DiscQuery	0	0	0	0
SMTP	Threshold	0	0	0-1	0-1
SMTP	Errmap	0	0	0+	0+
SMTP	UseBrackets	0	0	0-1	0

Allowed XML Elements in VrfyXML (Part 2)

Type	Element	POP3	DataBase	Multi	Sub-Verifier
Connect	Timeout	0-1	0-1	0-1	SAME, INHR
Connect	Credentials	0	0-1	0	SAME
Connect	BackendMax	0-1	0-1	0-1	SAME, INHR
Connect	BackendIdleTime	0-1	0-1	0-1	SAME, INHR
Connect	Host	1+	1+2	0	SAME
Connect	HostListOrder	0-1	0-1	0	SAME
Multi	Domain	0	0	0	0+
Event	InterEnumTime	0	0-1	0-1	0
Event	InterDiscTime	0	0	0-1	0
LDAP	BaseDN	0	0	0	SAME
LDAP	Canonical	0	0	0	SAME
LDAP	Filter	0	0	0	SAME
LDAP	StripPfx	0	0	0	SAME
LDAP	EnumFrom	0	0	0	SAME
LDAP	EnumTo	0	0	0	SAME
LDAP	EnumFilter	0	0	0	SAME
LDAP	EnumTest	0	0	0	SAME
LDAP	DiscFilter	0	0	0	SAME
LDAP	DiscAttr	0	0	0	SAME
DB	Vendor	0	1	0	SAME
DB	DBName	0	1	0	SAME
DB	VrfyQuery	0	1	0	SAME
DB	AuthQuery	0	0-1	0	SAME
DB	EnumQuery	0	0-1	0	SAME
DB	DiscQuery	0	0-1	0	SAME
SMTP	Threshold	0	0	0	SAME
SMTP	Errmap	0	0	0	SAME
SMTP	UseBrackets	0	0	0	SAME

