EMC RID Agent v1.0 Deployment Guide

Prepared by: John Field, EMC
Date: November 26, 2012

Table of Contents

Table of Figures

1. Introduction

As part of our support for the Enduring Security Framework effort, the EMC CTO Office is participating in a joint PoC (Proof of Concept) effort called ACIIS (Automated Cyber Incident and Indicator Sharing). At the PoC's conclusion, EMC will have demonstrated that we are capable of consuming and producing live structured cyber threat data with other participants following RFC 5070 (IODEF), RFC 6545 (RID), and RFC 6546 (RID Transport).

The goal of "Phase 0" of the PoC is to send and receive placeholder (mocked-up) cyber threat information with the other members participating in Phase 0. The goal of Phase 1 is to leverage work done in Phase 0, increasing the complexity of the exchanged data from notional, to a more realistic cyber defense scenario such as exchanging watchlist information. Finally, the planned goal of Phase 2 is to leverage the work done in Phases 0 and 1, ultimately being able to communicate live structured threat data.

This document provides the reader with the information needed to deploy and maintain an instance of the EMC RID Agent.

2. Deployment Environment

The minimum system requirements for hosting the EMC RID Agent implementation are as follows:

- Windows Server 2008 Service Pack 2
  - (VM deployment is fine, that's what we do).
- Dual core AMD Opteron™ processor 8214 (four processors)
- 32 bit architecture
- 16 GB Memory
- 40 GB HDD

The actual RID Agent install will require < 1GB of disk space.

For the POC, the volume of information being exchanged is minimal and so questions such as system capacity and raw processing speed are not an immediate concern. The EMC RID Agent runs well on any typically configured developer's desktop or laptop. Any modestly spec'd physical or virtual lab machine will be just fine for the purposes of the ACIIS POC deployment.

The complete software stack includes:

- Java JDK6.x
- Tomcat 6.x
- EMC Documentum XDB 10.x
  - This is a native XML Database for RID messages.
    - (Developer download of the executables and license are free with registration. See: https://community.emc.com/docs/DOC-3139).
  - Also includes EMC Calumet libraries

        - (An Xproc implementation from EMC that follows the W3C recommendation).
- The EMC RID Agent application
    - The actual distribution is a WAR file containing all required libraries.
        - Approx. 30 MB in size.

In addition, the deployment requires opening firewall port 4590 (inbound and outbound) for HTTPS. Note that only port 4590 is required in production. The deployment SHOULD be configured so that the Tomcat server that hosts the RID Agent is not reachable on 8080, or 8443, etc.

The following figure shows the current ACIIS Phase 0 POC deployment scenario.

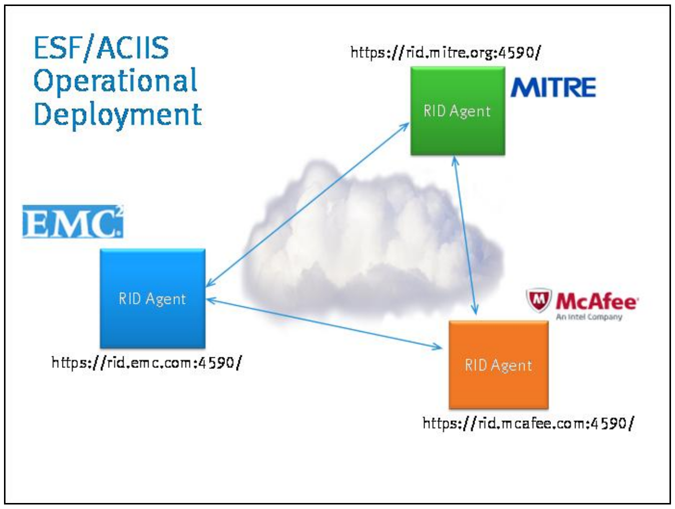

ESF/ACIIS Operational Deployment
https://rid.mitre.org:4590/
RID Agent
MITRE
EMC
RID Agent
https://rid.emc.com:4590/
McAfee
An Intel Company
RID Agent
https://rid.mcafee.com:4590/

Figure 1: Example ESF/ACIIS Operational Deployment.

The exact organizations that may participate in the ESF/ACIIS PoC information sharing network is subject to change; the specific participants depicted in Figure 1 were correct as of this writing. Additional ESF/ACIIS participants may be added in the future. It is important to note that some organizations (EMC included) may not provide DNS name resolution for hosts located in the DMZ, and so actual connections may be done using IP addresses rather than hostnames.

The functionality of the EMC RID Agent includes the capability to send and/or receive RID **Query** and RID **Report** messages.

When the EMC RID Agent receives a good RID Report or Query message, we'll send back an **Acknowledgement** message with <RequestStatus AuthorizationStatus="Pending"/>. If we receive a bad RID message (i.e. due to a schema violation) we'll return an Acknowledgement message with <RequestStatus AuthorizationStatus="Denied" Justification= "CannotProcess"/> and no <IncidentID>. If we have an internal failure of some description (e.g. we can't persist the message), then we'll return an Acknowledgement with <RequestStatus AuthorizationStatus="Denied" Justification="Other"/> without any <IncidentID>. Any other kind of inbound RID message will be echoed back to the sender, with a wrapper element that indicates the RID message type that we parsed (as a sort of confirmation of receipt) along with a 200 OK status. However, we won't actually persist the RID message unless it is a Report or a Query.

Support for XML Security (as per ACIIS POC Goal 5) is planned for a future release.

3. High Level Architecture

The RID Agent implementation has been deployed to the EMC DMZ and is visible from the public Internet. Any (appropriately authorized) counter-party that sends EMC a RID message at https://137.69.116.31:4590/ will receive a response as described above.

As shown in Figure 2, below, the system is protected by both an external and internal firewall. At the externally facing firewall only port 4590 is open for connections inbound, and outbound. In addition, we have implemented application-level white-list enforcement, and the counter-party must connect via HTTPS, and present a trusted PKI certificate. (Details of configuring the white-list and truststore are provided in section (TODO insert ref.), below.

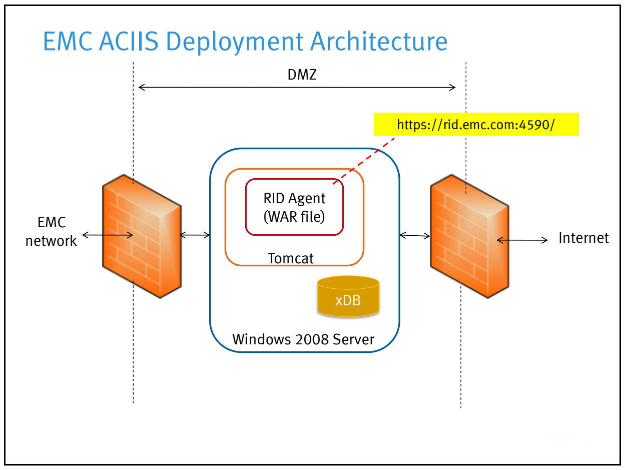


Figure 2: EMC RID Agent DMZ Deployment.

The implementation of the EMC RID Agent is essentially based on an "XRX" Architecture ("XForms, REST, XQuery"). This is, roughly speaking, a RESTful Java application stack. Of course because this is just a service oriented messaging agent, there is no UI to speak of, so there really aren't any XForms present in the core RID Agent application.

We use the Spring Framework MVC for handling the HTTP request/response aspects and we have Java code that invokes XProc/XQuery pipelining to do the actual processing on the RID/IODEF messages, both in-bound and out-bound. We store these messages in a native XML database called XDB. One can think of XDB as a kind of MySQL database, only without the SQL part. ☺ It is a native XML store that you write and query using the Xquery language.   It is well suited to this task because most of what we are doing here is persisting raw XML messages.

The following figure provides an overview of the component-level architecture.

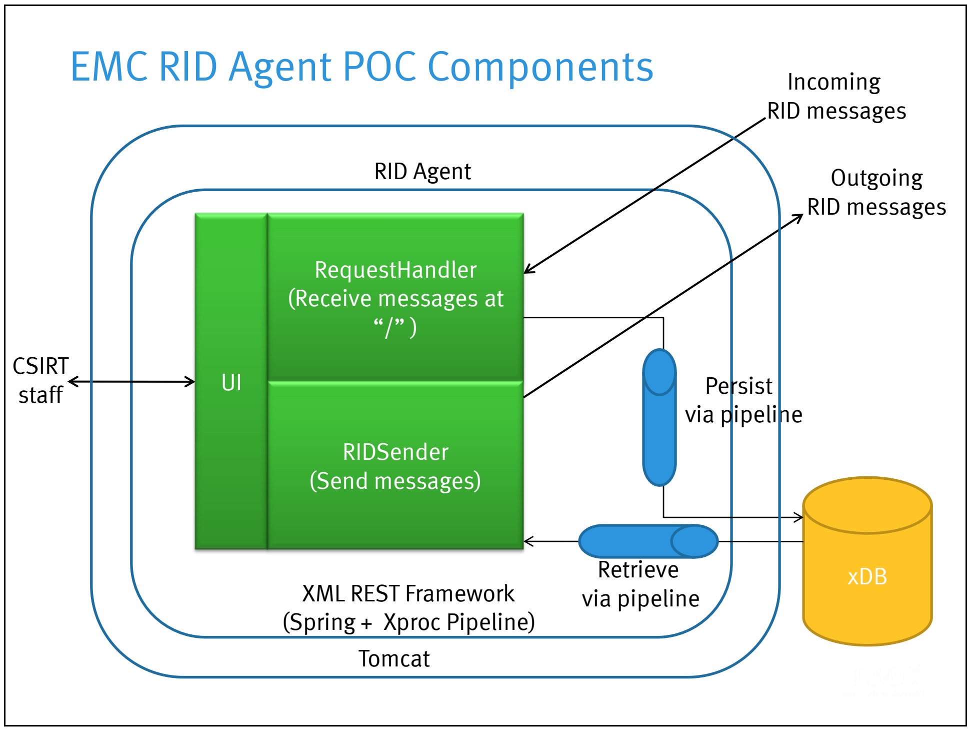


Figure 3: Component Level Architecture of the EMC RID Agent.

As shown in the above figure, the EMC RID Agent consists of two major subsystems. These are the sending function and the receiving function. The sending function is a simple Web application delivered via JSP pages. The receive function is a Spring MVC-based RESTful service (no UI) that is listening for POST operations on port 4590 for the root resource "/."

3.1. Core Design Principles

An important principle in the EMC RID Agent architecture and design is that we want to avoid lots of "no-value" object transformations in our XML message processing. That is, in a typical Java+XML+SQL application, there is a need to do lots of DOM or SAX processing of the inbound XML messages in order to deserialize the XML into Java objects, and then subsequently to move these Java objects into SQL tables via JDBC. In the send case, the reverse is true. The data would typically need to be read from a SQL database table(s) and then instantiated into Java objects via JDBC calls, and then finally serialized out as XML using DOM or SAX processing. This is a lot of "no value" code to write, and debug, and maintain which ultimately performs no meaningful business function, but rather implements internal technical manipulations. The actual business use case requires yet more code, and is usually implemented in a business service tier, which operates on the intermediate Java objects.

The approach we have taken avoids most of these no-value lines of code in the middle tier, by working with the XML in its native form, using the XProc programming language. The following sections will describe the theory of operation of the EMC RID Agent in more detail.

3.2. Inbound RID Message Processing

We accept the XML RID message inbound at a Spring MVC service endpoint, and then send it to a Java request handler class, which will immediately invoke an XProc processing pipleline called requestHandler.xpl. This main pipeline checks to be sure the message is schema valid, and if so, examines the actual RID message type, and then routes it accordingly. There are sub-pipelines that can be called from the main pipeline that are each dedicated to handling a specific operation. A RID message of type Query or Report is dispatched to a subpipeline called addResource.xpl. The RID Query message is ultimately handled via another call to an XQuery module called addQuery.xq. A Report message is handled by a similar call to a subpipline called addReport.xq.

The pipeline processing works directly on the native XML messages, and then ultimately stores the message, as native XML in the XDB database, using XQuery calls. Manipulations of the RID messages are done exclusively using the XProc programming language, which follows a declarative rather than a procedural programming paradigm. This approach to RID message processing avoids any procedural manuipulations written in Java in the middle (business) tier code.

3.2.1. Persistence

The XDB database library is named TAXII (historical). When a new message is received (and it is determined to be valid) the system generates a unique UUID for the message, and stores it in a sub-library in XDB that is named for that UUID. All RID Query documents are stored with the name query.xml. All RID Report documents are stored with the name report.xml. Uniqueness of the database entry is guaranteed by the sub-library (folder) name being chosen as a UUID.

3.2.2. Implementing Additional RID Message Types

Implementation of additional RID message types will require that the developer provide the appropriate sub-pipelines for, say, handling a RID TraceRequest, or RID InvestigationRequest, and so on. In the current implementation we have just stubbed these out. If we receive a message of a type other than RID Report or RID Query, we will wrap the message with an XML element identifying the type we parsed, and then we echo the message back to the sender, as a sort of return receipt. This behavior is not conformant with the RID RFC of course, but this stubbed out implementation is supplied as a way to indicate to senders that the received message is not supported, and also so that it will be clear to future developers where they will need to put their XProc code.

3.2.3. Support for RID Acknowledgement Messages

When the EMC RID Agent receives a good RID Report or Query message, it will send back an Acknowledgement message with <Request Status AuthorizationStatus="Pending"/> and our UUID as the IODEF <IncidentID>. If we receive a bad RID message (i.e. schema violation) we'll return an Acknowledgement message with <RequestStatus AuthorizationStatus="Denied" Justification= "CannotProcess"/> and no <IncidentID>. If we have an internal failure (e.g. can't persist the message), then we'll return an Acknowledgement with <RequestStatus AuthorizationStatus="Denied" Justification="Other"/> without any <IncidentID>. Any other kind of inbound RID message will be echoed back to the sender, with a wrapper element that indicates the RID message type that we parsed (as a sort of confirmation of receipt) along with a 200 OK status. However, the EMC RID Agent won't actually persist the RID message unless it is a Report or a Query.

The processing approach used in the requestHandler.xpl pipeline is to do a try/catch block that calls the schema validation sub-pipeline.

If this validation succeeds, then we enter the choose Xproc step (i.e., this is the Xproc equivalent of a "switch", or "case" statement) and look at the RID message type, and dispatch accordingly.

If the schema validation fails, then an exception is thrown, and we use an inline document to format a RID Acknowledgment message, with a <RequestStatus> element containing @AuthorizationStatus="Denied" @Justification="CannotProcess". We use a MsgType of "ext-value" as a mechanism to bypass the other processing steps in the requestHandler.xpl, and go to the "otherwise" path, which will do a substitution of MsgType to be "Acknowledgment" and then it returns this document on the primary output port.

The pipelines can be tested at the command line using Calumet by following the instructions in the Calumet user guide.   When doing this pay particular attention to having the right plug in support, i.e. the needed jars must be in the XDB /lib directory as described on page 36 of that guide.

3.3. Outbound RID Message Processing

The EMC RID Agent sending function implementation is a separate Web application called RIDSender. The user interface consists of a simple JSP form where the user must enter the UUID of the RID message to send (as stored in the XDB database) and the URL of the target RID agent. A user may create or edit a RID message using their favorite XML editor, and then add this into the XDB database manually by using the XDB Administrators tool. Once the message has been put in the XDB database, it will be available for the RIDSender to send.   The RIDSender implementation will read the XDB database using XQuery and retrieve the indicated message by the UUID. It will then send that message to the counterparty via HTTPS using a Java HTTP client library. Whatever HTTP response is received from the sender will be displayed in the user's browser.

3.4. No Incident Handling System Integration

It is important to note that the EMC RID Agent is a standalone deployment and at this time it has not actually been integrated with any back-end Incident Handling System. In a real deployment, there would need to be a provision to periodically move the inbound messages received by the RID Agent from the XDB database, into the associated Incident Handling System (e.g. inside the enterprise firewall). Once transferred to the IHS, the RID messages could be reviewed by a qualified cyber security analyst. The analyst would then make a determination as to how to triage and process the messages. Similarly, for an outbound message, the cyber security analyst would likely be working at a secure Incident Handling System – which is a separate system from the EMC RID Agent – and the queue of requests that are created in the IHS would then need to be transferred (periodically, or on demand) to the EMC RID Agent database for delivery to the sharing peer(s).

This IHS integration is necessarily site specific, and so is not included in the scope of the current implementation.   This integration may be scoped to be part of a future ACIIS PoC phase.

4. Required Skill Set for EMC RID Agent Implementation

Adopters of the EMC RID Agent who plan to modify the open source implementation should be skilled in Web Application deployment using Apache Tomcat, the Java programming language, the Spring Framework, and authoring Xproc processing pipelines. We use the Spring Source Tools Suite (Eclipse based) as an IDE. Most of our lower-level plumbing implementation is achieved via libraries like Spring Framework, Spring Security, and Spring MVC, and so the Agent programming required is limited to the "business" logic of handling the RID messages and responses. As noted, this is done primarily in the XProc programming language, following the W3C Recommendation for Xproc. For more information, see: http://www.w3.org/TR/xproc/ .

As an adopter, if you plan only on deploying and running the Agent, and do not plan on modifying the software, then knowledge of Spring Framework and Xproc are not required. One may successfully deploy the system with knowledge of Tomcat, your favorite editor, and standard TLS setup operations using JDK keytool.

Full disclosure: our application stack was chosen as being REST-friendly because strategically that's where we expect this effort to go in the longer run. If we really wanted to just stick strictly with RFC 6545 and 6546 for the long haul, then having Spring MVC as our core framework may be considered overkill…but as described in the ROLIE draft, another option is that the strategic direction is moving towards leveraging additional URLs beyond just "/" and so in the long run the EMC RID Agent architecture is positioned to do just that.

5. Installation and Configuration

This section provides more detailed configuration instructions for deployers who intend to install an instance of the EMC RID Agent.

We have asked our local firewall administrator to open the required address and port for any peers. As of this writing McAfee, MITRE, IBM, Boeing, HP, and Symantec are authorized and permitted to connect. In order to authorize any additional participants it will be necessary to open a support case with EMC IT Security administration. The firewall rules maintenance is a prerequisite for any adopter and will affect both the new participant, as well as all existing participants.

Because this is a PoC implementation, there is no single-button installation wizard for the EMC RID Agent. The operator must install and configure the following prerequisites manually on a Windows Server 2008 system:

1. Install Java JDK 6.x
2. Install XDB 10.x
3. Install Tomcat 6.x

Once this has been done successfully, you may proceed to the next steps. Note that the EMC RID Agent has not (yet) been tested on Linux. While we believe it should work fine in a Linux-based deployment, that testing was not in scope for Phase 0 of the POC.

As discussed, the EMC RID Agent application itself is simply a Web Archive (war) file. Essentially, it can be deployed by dropping it into the webapps directory of a running Tomcat instance. It will then be automatically deployed, and will create a connection to the backend XDB database (that should be running on the same localhost). The RID Agent application will then attempt to listen for connection requests, and process any received RID messages.

A number of individual configuration files must be edited in order to customize the deployment and establish the new instance as a unique, trusted RID Agent. These configuration tasks include for example editing the certificate trust store(s) and the RID Agent white-list(s) settings that govern who is allowed to connect to this RID Agent.

Contact EMC to obtain a copy of the EMC RID Agent distribution kit (or download from the EMC Developer Network Web site: https://community.emc.com/to-be-supplied....), and using your favorite editor, prepare to edit the configuration details in the following files:

- <TOMCAT_HOME>/conf/server.xml
- <DIST _BASE>/WEB-INF/RIDSystemServlet.xml
- <DIST_BASE>/WEB-INF/applicationContext.xml

The Tomcat customizations are done only once, as part of your initial Tomcat installation.   When deploying a new version of the EMC RID Agent (i.e. a new release or upgrade) it is not anticipated that the server.xml file settings would need to change. However, the EMC RID Agent application-specific configurations would need to be repeated anytime there were a new EMC RID Agent distribution released.   One proven method to maintain war file customizations such as this would be to create a Maven “overlay project.” The Maven overlay project would have a dependency on the EMC RID Agent war file and would “overlay” the site-specific configuration files in the WEB-INF directory. Whenever a new war file is released, one may essentially run the overlay build, which would unpack and then rebundle that war file distribution replacing the generic configuration files included in the distribution with the corresponding site-specific custom configuration files.   Details of how to do a Maven overlay build in this way can be found on the Apache Maven Web site.

5.1. Configure the RID Agent Port

This step describes how to configure the application to use port 4590 and be the root application so that requests can be directed to http://hostname:4590/

1. Modify the server.xml found in your TOMCAT_HOME/conf folder by adding the following section. Refer to the Tomcat docs on the Apache Web site for the meaning of these elements.

```
<Service name="TaxiiService">
 <Connector port="4590" protocol="HTTP/1.1"
                   connectionTimeout="20000"
           redirectPort="9443" />
 <Engine name="TaxiiEngine" defaultHost="localhost">
        <Host name="localhost"  appBase="taxiiwebapps" unpackWARs="true"
                 autoDeploy="true" xmlValidation="false" xmlNamespaceAware="false">
                  <Context path="" docBase="TAXII" debug="0" reloadable="true" />
        </Host>
 </Engine>
</Service>
```

2. Create a folder called "taxiiwebapps" under TOMCAT_HOME and place the application WAR file there.
3. Note that in the <Context> element setting the path="" will ensure that the app is picked up as the Root app i.e. at "http(s)://hostname:4590/".   (Note: Currently one can also access the application at http://hostname:4590/TAXII/, this seems to be the way Tomcat sets it up by default. Effort is needed to turn this off).
4. When you run Tomcat it automatically creates a folder "TaxiiEngine" and it's subdirectory "localhost" under TOMCAT_HOME/conf/.   It also creates a similar structure under TOMCAT_HOME/work/.
5. Optional: If you want to deploy the "manager" and "host-manager" applications (that come default with Tomcat) for the TaxiiEngine also, then copy the folders "manager" and "host-manager" from webapps to taxiiwebapps.
    a. Note that this also gets rid of the warning when you run Tomcat.
    b. Also copy the files manager.xml and host-manager.xml from the TOMCAT_HOME/conf/Catalina/localhost to the folder TOMCAT_HOME/conf/TaxiiEngine/localhost.
        i. Some Tomcat docs specify this step whereas others say this is optional.

5.2. Configuring TLS for the application

As per RFC 6546 use of TLS is required with RID. The steps to configure TLS for Tomcat are not unique to the EMC RID Agent, but would be the same for any TLS-enabled application.   To enable TLS on your RID System two things must be done:

1. Install your server certificate in a keystore reachable by the Tomcat runtime.
2. Configure Tomcat to point to that keystore.

To do this, edit the server.xml found in your TOMCAT_HOME/conf, and add the following snippet:

```
<!-- SECOND CONFIGURATION TO RUN TAXII ON PORT 4590 -->
 <Service name="TaxiiService">
  <Connector port="4590" protocol="HTTP/1.1"
```

```
          SSLEnabled="true"
      maxThreads="150" scheme="https" secure="true"
      enableLookups="false" disableUploadTimeout="true"
      acceptCount="100" minSpareThreads="25" maxSpareThreads="75"
      clientAuth="true" sslProtocol="TLS"
      connectionTimeout="20000"
      redirectPort="9443"
      keystoreFile="{C:\path to the aciisKeystore}"
      keystorePass="password goes here:, i.e. changeit"
          truststoreFile="{C:\Path to aciisKeyStore i.e., JDK path, …jre\lib\security\cacerts}" />
  <Engine name="TaxiiEngine" defaultHost="localhost">
    <Host name="localhost"  appBase="taxiiwebapps" unpackWARs="true"
          autoDeploy="true" xmlValidation="false" xmlNamespaceAware="false">
   <Context path="" docBase="TAXII" debug="0" reloadable="true" />
    </Host>
  </Engine>
</Service>
```

5.3. Enable TLS Trust Relationships

In this step, you are configuring your truststore to recognize the certificates that will be presented by your peers when they connect.   Install the CA certificate (or the chain of certificates to the CA, if available) of each RID peer in your information sharing group into your truststore.   If the peer server certificate is self-signed then install that certificate directly in the truststore. You may do this using JDK keytool import command.

Also, configure Tomcat to point to the truststore, as shown in the code snippet above.

5.4. Configuring TLS Client Certificate Authentication

When our RID system acts as a client of a peer RID system (i.e. the sending subsystem), the client certificate sent will be our server certificate used for TLS.   This certificate should be in your keystore so that you can send it when it is solicited by the server in the TLS negotiation.   Tomcat should point to the appropriate keystore as in the above step.

When receiving a client request, our RID system will validate the client certificate presented. To do so, install the CA certificate of any RID peer which sends its client certificate in your truststore. Again, this is done using JDK keytool.

Tomcat should point to the appropriate truststore as shown in the above step.

5.5. Configuring xDB Connectivity

Before you deploy the war file, you must configure XDB. Using the XDB Administration tool, create a database in XDB called “TAXII.” Choose a username and password for the TAXII database administrator account, and record these for later use.

Set the xDB connection password in the /WEB-INF/RIDSystem-servlet.xml file to be the password for the "TAXII" database chosen in your xDB installation (N.B., The name “TAXII” is historical. The name of the project was changed after development began).

Search for the following XML snippet and replace with the appropriate password:

```
<bean id="xprocPool" class="com.emc.cto.xproc.xdbplugin.XDBSessionedXProcPool">

  .....

  <constructor-arg index="4"><value>REPLACE_HERE</value></constructor-arg>

  ......

</bean>
```

5.6. Removing the Log4j Warnings (OPTIONAL)

Place the setenv.bat file in the TOMCAT_HOME/bin location. This will be picked up by the default startup script.

The file contains details about how the contents are used, briefly this is used to avoid the log4j warning from appearing. This step is for convenience in reading the logs, but is not required.

5.7. Configuring the Whitelist

Only counter parties that are authorized may connect to the EMC RID Agent. The "cn" from the peer's PKI certificate must be added to the EMC RID Agent whitelist.

The whitelist is kept, and checked, in two places.

At the entry point, Spring Security enforcement is used. The spring security framework is providing coarse-grained enforcement. E.g. this enforcement determines whether the user can make a POST request to "/", or not.

Later, when the inbound RID message is passed into the EMC RID Agent application code, a second check is done. This second check is mostly a stub for future expansion, for when we may want to do enforcements based on the RID message type. As noted, this part is just stubbed out for now, so the check is just based on name (cn), not yet the name and message type. But it is likely that in the future we will need the capability to control who can send a TraceRequest, or an InvestigationRequest, and so on.

The whitelist that is enforced by the Spring Security configuration relies on the Spring userDetails service. Edit the userDetails bean in the /WEB-INF/applicationContext.xml file to add or remove additional parties to the authorized peer list. If the user is not configured in spring security they will get an HTTP status code of 403 forbidden.

Edit the file /WEB-INF/RIDSystem-servlet.XML to add or remove the appropriate users from the application-level whitelist. The whitelist bean uses the Spring "util:" name space and this bean is instantiated at runtime as a java.util.Set. If the requester "cn" is not in the set of users in this whitelist bean, they will get an HTTP status code of 404 not found.

5.8. Configuring Schema Validation

As of the v1.0 release, the EMC RID Agent is supporting XML schema validation. The first step in the processing pipeline on an in-bound RID message is to validate the message using the Xproc validate-with-xml-schema processing step.

The required XSD files include IODEF, RID, and XML Digital Signature:

- iodef-1.0.xsd
- iodef-rid-2.0.xsd
- xmldsig-core-schema.xsd

The required schemas are kept in the XDB, and at runtime are expected to be found in a sub-library called "/TAXII/schemas." These three XSD files are included in the war file distribution, under /WEB-INF/classes. These must be copied into the XDB database, as described below.

5.8.1. Note on Schema Validation in Production

The following code snippet, which appears at the beginning of the file for the xmldsig-core-schema.xsd, has been removed from the copy of the schema document that is actually stored in XDB. The reason is that our production XDB server is in the DMZ and it cannot access resources at arbitrary locations on the Internet (the DMZ firewalls prevent this). It seems that when you try to save this document into XDB, the XDB engine parses the schema file in order to validate it, and this causes a timeout because it seems to be chasing the reference to the DTD and this can't succeed on the DMZ machine due to the firewall rules. Removing this snippet works around that issue.

```
<!DOCTYPE schema
  PUBLIC "-//W3C//DTD XMLSchema 200102//EN" "http://www.w3.org/2001/XMLSchema.dtd"
 [
   <!ATTLIST schema
     xmlns:ds CDATA #FIXED "http://www.w3.org/2000/09/xmldsig#">
   <!ENTITY dsig 'http://www.w3.org/2000/09/xmldsig#'>
   <!ENTITY % p ''>
   <!ENTITY % s ''>
  ]>
```

Another alternative might be to load all the bootstrap data needed for the XDB database via an export/import approach, i.e. loading a template database after XDB has been installed. This method might also avoid the need to validate the schema document as it is saved into the xdb server, but this approach has not yet been tried.

5.9. Update the /etc/hosts File (OPTIONAL)

The EMC RID Agent depends upon DNS name resolution services. If DNS services are not available in your deployment environment, you must update the /etc/hosts file on the localhost. Edit this file to contain the hostnames and IP addresses of any peer RID system with which you plan to communicate. On the Windows Server 2008 system that EMC uses in production, this file can be found in: C:\Windows\System32\drivers\etc\hosts.

Update this file to contain the required RID peers, as shown below:

```
# Copyright (c) 1993-2006 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

127.0.0.1       localhost
::1             localhost


137.69.116.31     rid.emc.com
66.170.227.81     rid.mitre.org
170.225.96.254   VHOST0254.DC1.CO.US.COMPUTE.IHOST.COM     #IBM
156.139.16.49      aciis.hpl.hp.com
184.191.33.205   taxii.symantec.com
```

Afer the appropriate updateds have been done, simply save and close the file.

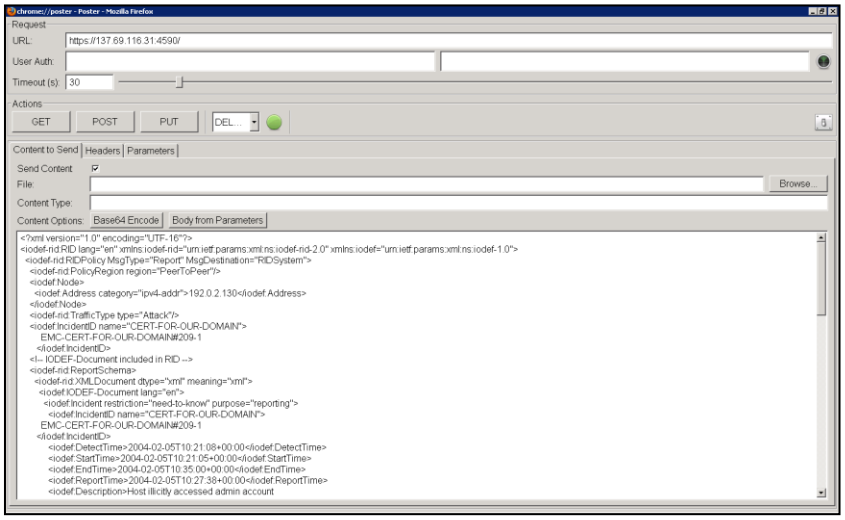
5.10. Restart Tomcat

Once all of the above configurations have been completed, you must redeploy the war file and then re-start Tomcat for the changes to take effect. Once that is done, you should be able to send and receive RID Query messages or Report messages as needed.

6. Installation Verification Procedure

Once your EMC RID Agent is successfully deployed you should be able to send it a valid RID message using any HTTP RESTful client tool, such as Firefox with the Poster plug-in.

Point your client at https://<hostname-or-IP-address>:4590/ and do an HTTP POST operation:

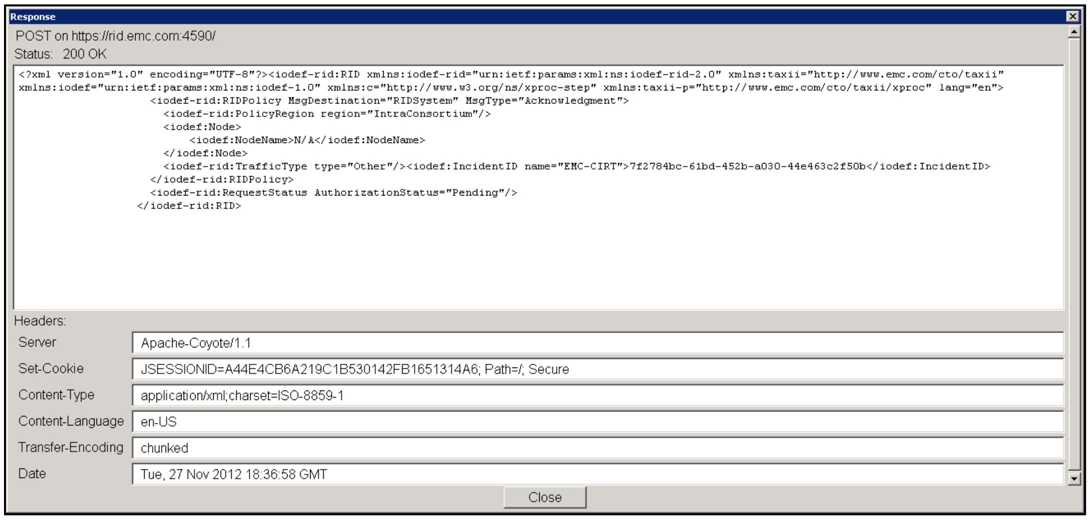


URL: https://137.69.116.31:4590/
User Auth:
Timeout (s): 30
Send Content ☑
File:
Content Type:
Content Options:

```
<?xml version="1.0" encoding="UTF-16"?>
<iodef-rid:RID lang="en" xmlns:iodef-rid="urn:ietf:params:xml:ns:iodef-rid-2.0" xmlns:iodef="urn:ietf:params:xml:ns:iodef-1.0">
 <iodef-rid:RIDPolicy MsgType="Report" MsgDestination="RIDSystem">
  <iodef-rid:PolicyRegion region="PeerToPeer"/>
  <iodef:Node>
   <iodef:Address category="ipv4-addr">192.0.2.130</iodef:Address>
  </iodef:Node>
  <iodef-rid:TrafficType type="Attack"/>
  <iodef:IncidentID name="CERT-FOR-OUR-DOMAIN">
    EMC-CERT-FOR-OUR-DOMAIN#209-1
   </iodef:IncidentID>
  <!-- IODEF-Document included in RID -->
  <iodef-rid:ReportSchema>
   <iodef-rid:XMLDocument dtype="xml" meaning="xml">
    <iodef:IODEF-Document lang="en">
     <iodef:Incident restriction="need-to-know" purpose="reporting">
      <iodef:IncidentID name="CERT-FOR-OUR-DOMAIN">
    EMC-CERT-FOR-OUR-DOMAIN#209-1
   </iodef:IncidentID>
      <iodef:DetectTime>2004-02-05T10:21:08+00:00</iodef:DetectTime>
      <iodef:StartTime>2004-02-05T10:21:05+00:00</iodef:StartTime>
      <iodef:EndTime>2004-02-05T10:35:00+00:00</iodef:EndTime>
      <iodef:ReportTime>2004-02-05T10:27:38+00:00</iodef:ReportTime>
      <iodef:Description>Host illicitly accessed admin account
```

Figure 4: Testing the EMC RID Agent using Poster REST Client Tool.

You should be asked to present your client certificate, and then you can complete the POST. The response to a RID Query or RID Report will be an Ackowledgement message with an HTTP status of 200 OK, as shown below:



POST on https://rid.emc.com:4590/
Status: 200 OK

```
<?xml version="1.0" encoding="UTF-8"?><iodef-rid:RID xmlns:iodef-rid="urn:ietf:params:xml:ns:iodef-rid-2.0" xmlns:taxii="http://www.emc.com/cto/taxii"
xmlns:iodef="urn:ietf:params:xml:ns:iodef-1.0" xmlns:c="http://www.w3.org/ns/xproc-step" xmlns:taxii-p="http://www.emc.com/cto/taxii/xproc" lang="en">
                    <iodef-rid:RIDPolicy MsgDestination="RIDSystem" MsgType="Acknowledgment">
                      <iodef-rid:PolicyRegion region="IntraConsortium"/>
                      <iodef:Node>
                          <iodef:NodeName>N/A</iodef:NodeName>
                      </iodef:Node>
                      <iodef-rid:TrafficType type="Other"/><iodef:IncidentID name="EMC-CIRT">7f2784bc-61bd-452b-a030-44e463c2f50b</iodef:IncidentID>
                    </iodef-rid:RIDPolicy>
                    <iodef-rid:RequestStatus AuthorizationStatus="Pending"/>
                  </iodef-rid:RID>
```

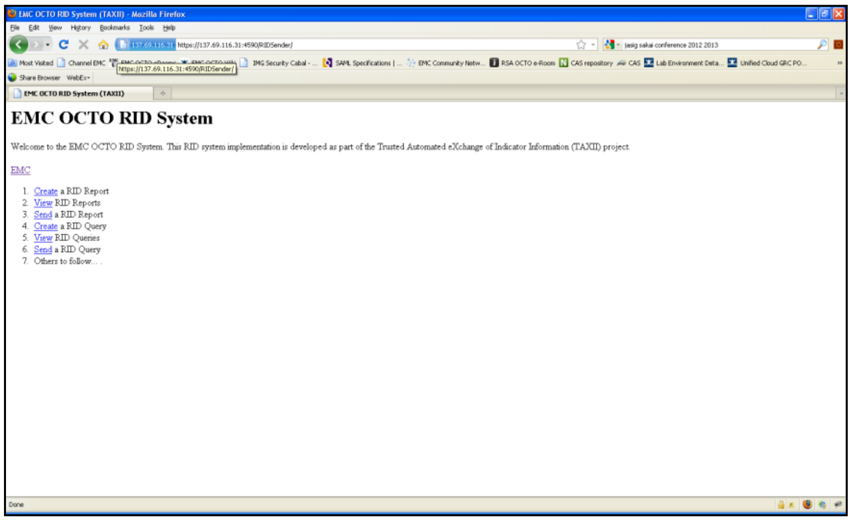
Headers:

Server	Apache-Coyote/1.1
Set-Cookie	JSESSIONID=A44E4CB6A219C1B530142FB1651314A6; Path=/; Secure
Content-Type	application/xml;charset=ISO-8859-1
Content-Language	en-US
Transfer-Encoding	chunked
Date	Tue, 27 Nov 2012 18:36:58 GMT

Figure 5: Successful RID Ackowledgement message as Received in Response to a RID Report.

To verify that the RID Sender application is running, point your browser at https://<hostname-or-IP-address>:4590/RIDSender.

The response should be a page that looks like the following:


EMC OCTO RID System (TAXII) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://137.69.116.31:4590/RIDSender/
EMC OCTO RID System (TAXII)
Done

EMC OCTO RID System

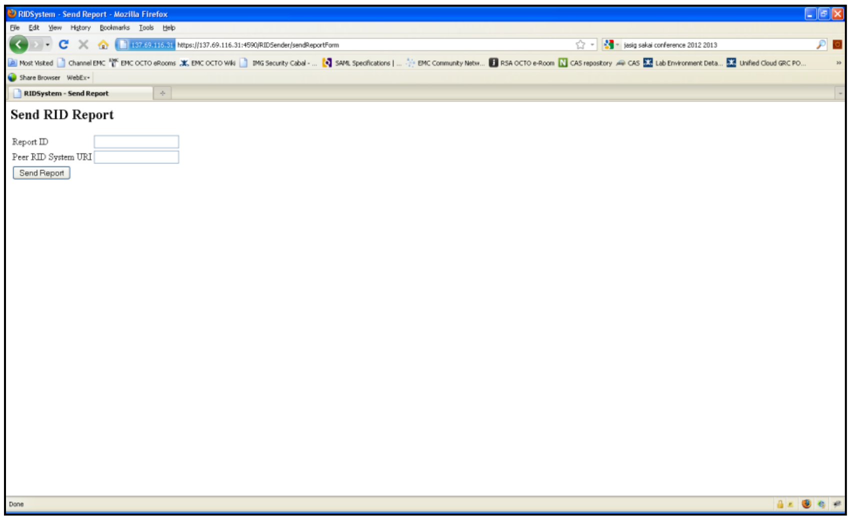
Welcome to the EMC OCTO RID System. This RID system implementation is developed as part of the Trusted Automated eXchange of Indicator Information (TAXII) project.

EMC

1. Create a RID Report
2. View RID Reports
3. Send a RID Report
4. Create a RID Query
5. View RID Queries
6. Send a RID Query
7. Others to follow… .

Figure 6: EMC RIDSender Home Page.

Clicking on the link to send a RID Report should result in the following page:


RIDSystem - Send Report - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://137.69.116.31:4590/RIDSender/sendReportForm
RIDSystem - Send Report
Done

Send RID Report

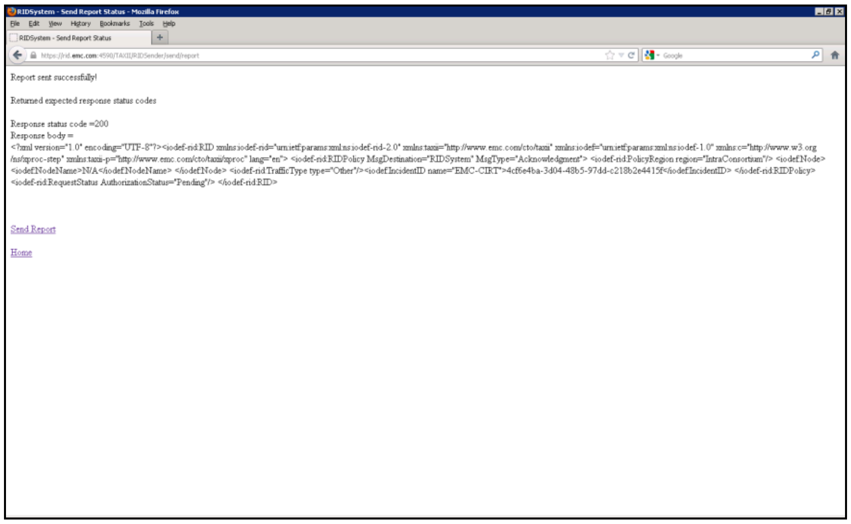
Report ID	
Peer RID System URI	

Send Report

Figure 7: EMC RIDSender Send RID Report Page.

From this point, you may provide a valid UUID and target URL to send a RID Report. Similarly for RID Query messages, just use the Send RID Query link from the RID Sender home page.

Finally, the result page from a successful send will include the RID Acknowledgement message, as shown below:


RIDSystem - Send Report Status - Mozilla Firefox
File Edit View History Bookmarks Tools Help
RIDSystem - Send Report Status
https://rid.emc.com:4590/TAXII/RIDSender/send/report
Google
Send Report
Home

```
Report sent successfully!

Returned expected response status codes

Response status code =200
Response body =
<?xml version="1.0" encoding="UTF-8"?><iodef-rid:RID xmlns:iodef-rid="urn:ietf:params:xml:ns:iodef-rid-2.0" xmlns:taxii="http://www.emc.com/cto/taxii" xmlns:iodef="urn:ietf:params:xml:ns:iodef-1.0" xmlns:c="http://www.w3.org
/ns/xproc-step" xmlns:taxii-p="http://www.emc.com/cto/taxii/xproc" lang="en"> <iodef-rid:RIDPolicy MsgDestination="RIDSystem" MsgType="Acknowledgment"> <iodef-rid:PolicyRegion region="IntraConsortium"/> <iodef:Node>
<iodef:NodeName>N/A</iodef:NodeName> </iodef:Node> <iodef-rid:TrafficType type="Other"/><iodef:IncidentID name="EMC-CIRT">4cf6e4ba-3d04-48b5-97dd-c218b2e4415f</iodef:IncidentID> </iodef-rid:RIDPolicy>
<iodef-rid:RequestStatus AuthorizationStatus="Pending"/> </iodef-rid:RID>
```

Figure 8: RID Acknowledgement Message Received in Response to a Successful Send.

If each of these tests has passed then the EMC RID Agent has been successfully installed.