Threat Analysis of Recursive Inter-Network Architecture Distributed Inter-process Communication Facilities

Jeremiah Small, John Day, Lou Chitkushev
Computer Science Department
Boston University Metropolitan College
{jdsmall, day, ltc}@bu.edu

***Abstract*—In the book *Patterns in Network Architecture: A Return to Fundamentals*, a new approach to designing and building networks based on the fundamental concepts of Interprocess Communication (IPC) is discussed.[1] This model departs from the traditional concept of a layered architecture in which each layer has a distinct and different purpose, and instead uses repeating layers of common functions optimized to a specific range of bandwidth, QoS, and scale. From these design principles an elegant architecture has been proposed called Recursive Inter-Network Architecture (RINA).[2][6][7][8][9] Earlier papers have discussed how the RINA design inherently resists some of the security issues that plague the TCP/IP architecture.[3] In this paper we will explore the security of RINA's equivalent to a network layer, a Distributed IPC Facility (DIF). A DIF is the basic building block of RINA networks. We will examine the threat model of the RINA API and Common Distributed Application Protocol (CDAP) to show that a DIF is a securable container.**

I. What is a DIF?

Recursive Inter-Network Architecture (RINA) networks consist of layers of building blocks called distributed interprocess-communication facilities or "DIFs". This paper will endeavor to show that from and architectural perspective RINA's equivalent of a network layer, a DIF, is a securable container. As a disclaimer, we note that the proceeding analysis persumes that the host operating system on which the RINA implementation is running has not been compromised. If processes outside of the RINA implementation can read the privileged memory of an IPC process, then all bets are off.

Each DIF consists of one or more IPC processes containing the following components: a Resource Information Base (RIB), IPC Management Tasks (Enrollment, Routing, Directory, Resource Allocation, Security Management), Relaying and Multiplexing Task (RMT), a Service Data Unit (SDU) Delimiter task, an SDU protection task, and the Error and Flow-Control Protocol (EFCP) machine. Each IPC process may also have one or more Application Processes (AP) running "above" the DIF that will rely on the DIF to communicate with other APs that have access to the same DIF. The IPC Process itself is an AP and relies on DIFs "below" it for communication and so on down to the physical transmission medium.

No security audit would be complete without discussing the assets involved. In this case we are discussing a processing system. Assets include: User data, Management Data, and computing resources. Each of these may sustain attacks against them in the form of interruption, interception, modification, or fabrication. User data is any data which is stored on or transmitted to or from a computing system, or that can be inferred from such data. Examples include: documents, media files, source code, or account balances among others. Management data is a special subset of user data that pertains to the operation of the computing system or network. Examples include: real-time performance data, names, addresses, cipher keys, and policy data among others. Computing resources refers to the computing systems themselves. Blocking access to, remotely controlling, and or electronically or physically disabling these resources are examples of attacks against them.

When exchanging messages, a DIF is similar to a TCP/IP layer in that the protocol machine on one processing system (in this case an IPC process) communicates only with protocol machines in the same layer (DIF), usually on other processing systems. For example in TCP/IP, TCP protocol machines "talk" to other TCP protocol machines on other processing systems, but not to IP protocol machines. Similarly IP only "talks" to IP. This aspect is the same in RINA in that IPC processes in a DIF only communicate with other IPC process in the same DIF and only within the same processing system. The messages that get passed between IPC processes are called Protocol Data Units (PDUs), these consist of Protocol Control Information (PCI)[1] and a payload which RINA calls an SDU. An SDU may contain one or more complete PDUs or a fragment of a PDU from the layer above. Operationally, this distinction may not matter much because of RINA's inherent recursiveness, however it is useful to make a distinction when considering the security between layers. For the remainder of this paper we will use the term PDU to refer to an *(n+1)-DIF* SDU with *n-DIF* PCI.

II. Is a DIF a Securable Container?

A. What is a securable container?

To answer the question: *Is a DIF a Securable Container?* we need to first define what a Securable Container is. The following definitions are from the New Oxford American Dictionary. [4]

[1] sometime referred to as header and trailer

Secure:
not subject to threat; certain to remain or continue safe and unharmed; protected against attack or other criminal activity

Securable:
Able to be secured.

Container:
an object that can be used to hold or transport something

Attack:
take aggressive action against

Threat:
a person or thing likely to cause damage or danger

The reader should note that in our context, "attack" can be either active or passive. An example of an active attack might be attempting to gain unauthorized access to the container. An example of a passive attack might be eavesdropping on a conversation.

From these definitions and our context we can derive the following definition.

Securable Container:
A structure used to hold or transport something that can be made to be not subject to threat.

In order to test if this definition can be applied to a DIF we will need to answer the following additional questions:

1) What does it mean to attack a DIF?
2) What information would an attacker need in order to mount an attack on a DIF?
3) How could an attacker get the required information to attack a DIF?

III. What Does it Mean to Attack a DIF?

A DIF is a logical container that holds information about a collection of network nodes and facilitates communication between them. As we have already established, to attack something is to take aggressive action against it. This implies that to attack a DIF one would have to attack the data stored on one or more of the nodes, the communication between the nodes, or else the operation of the nodes themselves. Attacking could also include joining the DIF with a false positive authentication or without a proper authentication at all. We must also consider the varying scope with which a DIF could be attacked. The target of an attack could be a specific node within the DIF, any single node within the DIF, or else the collection of all the nodes in the DIF.

A. Traffic Attacks

In network communication, there are 4 primary types of attacks:[10]

Interruption:
Prevent PDUs from reaching their intended destination

Interception:
Read PDUs as they traverse the network without alerting the source or destination

Fabrication:
Generate and send fake PDUs to the destination that look like they came from a different source

Modification:
Changing the content of a PDU while in transit between the sender and receiver

B. Attacks on DIF Members

Now let us consider attacks on the DIF members themselves. Each DIF member stores the following information in its RIB. Any or all of these could be the target of an attack.

a) Forwarding Table Data
b) AP Synonyms
c) AP Addresses
d) Whatevercast names/synonyms
e) Inter-DIF Directory entries
f) List of DIF Members
g) Key material
h) Address Allotments
i) Flows
j) Authentication Credentials
k) Authorization Data (Access Control Rules)
l) Logs
m) Performance Metrics
n) Policy Data

1) Attacks on a Specific DIF Member: Let us consider the possible attacks against a specific DIF member. An attacker may wish to take the node off-line or otherwise disable it completely. They may wish to masquerade as this node in order to send fabricated traffic or use it to elevate their privilege by using a more trusted node than the one they have direct access to. They may simply wish to intercept the traffic of this specific node to eavesdrop on conversations of a specific user or set of users. They may wish to gain access to the specific system in order to gather information or execute their own code. They may also wish to disrupt communication by affecting the network's ability to provide the requested Quality of Service (QoS).

Motivation for attacking a specific DIF member may be to gather specific targeted information, execute a Denial-of-service (DoS) attack on a specific service, or compromise the system to use as a stage for further attacks among other reasons.

2) Attacks on Any DIF Member: Similar to an attack on a specific DIF member, an attacker may wish to target a DIF by attempting to attack *any* of its members. It could do this by intercepting any data being transmitted, or attempting to interrupt any random target.

An attacker may wish to attack any member indiscriminately in order to gain additional information about the DIF and its members which in turn may enable them to launch a more targeted attack. If the attacker can locate a less protected DIF member, it may be able to use that node as a staging ground from within the DIF to attack other members that may be better protected from non-member nodes.

3) Attacks on All DIF Members: Finally we come to attacks on the entire DIF, that is attacks against all of the members of the DIF at the same time[2]. In this case, an attacker may attempt to corrupt routing and forwarding data to affect data flow within the DIF. They may wish to disrupt or jam *all* traffic within the DIF. They may also wish to intercept and possibly record all traffic flowing across a DIF. They may also be attempting to use the DIF mechanisms to take down all of the member nodes causing a DIF-wide DoS.

Motivation for this may be to take out an entire subnet. Depending on the confidentiality policies being employed, an attacker may attempt to use the traffic within the DIF to compromise nodes in a DIF above it.

C. Other General Attacks

There are a few attacks against the DIF as a whole. These include escalation of privilege on access to data in the distributed RIB whether it be from a DIF member or non-DIF member. Finally gaining unauthorized access to the DIF (e.g. joining the DIF) including false positive authentication or circumventing authentication completely.

Motivation for these attacks would be to gather critical information such as key material or other member data to aid in more targeted attacks or to gain access to user data transmitted by this DIF.

Next we will need to discuss what information an attacker may need to perpetrate any of the aforementioned attacks.

IV. What Information Does an Attacker Need to Attack a DIF?

In this section we will discuss what information would be needed in order to launch one of the attacks just discussed. We will break this down into each of the attack scopes. In general, the following information will be needed:

a) DIF Name
b) specific member name/address
c) authentication credentials (even if these are null)
d) key material (for HMAC, cipher, etc)
e) SDU protection policy

A. Specific DIF Member

In order to launch any of the attacks discussed in section III-B1, the attacker must first be able to send network traffic to the target machine.

In order to intercept, modify, or fabricate network traffic to/from a specific DIF member, one would have to be able to identify the member in some way. This means that an identifier for the DIF itself as well as the node/AP in question would have to be known to the attacker[3]. An alternate way to get PDUs to a specific target is via an intermediate hop. In this case, the attacker would have to know what route PDUs will take through the DIF. However, before one could intercept or modify a PDU, the attacker would have to put themselves (or a proxy) along the route connecting source and destination APs. This requires that they know at least one, but likely multiple routes between the source and destination. To interrupt the traffic of one specific DIF member without affecting the traffic of other nodes, the attacker would have to identify the member. If the attacker is not concerned with interrupting traffic of other members, then less information may be needed.

In order to attack traffic between specific APs, the attacker would also have to know or be able to predict the connection identifiers of the traffic in question. PNA tells us in chapter 7 that ”connection identifiers will commonly be formed by the concatenation of the port-ids associated with [a] flow by the source and destination [Error flow control protocol machines]”[1]. Given this, if an attacker only wishes to capture traffic from one or more instances of the AP under attack, then they will have to discover at least the set of local port ids being used for that AP. If the attacker is interested in the traffic between two specific nodes, then they will also have to discover the port id being used for the other AP by the remote IPC process.

In addition to the network identifiers, the attacker will have to know (or be able to otherwise circumvent) the authentication credentials[4], key material, and SDU protection policy to be able to decode the traffic.

B. Any DIF Member

The information required to attack any random DIF member is a degenerate case of attacking a specific member. Like the specific DIF case, attacking any random DIF member will require discovering the identifier or address of a member, but in this case an anycast or broadcast synonym may be sufficient.

C. All DIF Members

Direct attacks against DIF traffic in the same manner as described in the preceding sections is also potentially possible using broadcast or multicast synonyms. Section III-B3 describes an attack against the forwarding tables, this implies being able to either compromise a host and RIB demon, or else intercept and modify or directly fabricate routing update messages.

D. DIF as a Whole

To intercept all traffic passing through the DIF, an attacker would require access to a number of DIF nodes sufficient to see all PDUs. Similarly, to disrupt all traffic in the DIF, an attacker would need access to enough DIF nodes such that all paths through the DIF pass through those attacker controlled systems.[5]

In order to join the DIF, a rogue AP would have to discover access credentials or otherwise circumvent the authentication sequence.

[2]or over time

[3]This also implies that the attacker must know the name space from whence these names come. Unlike their counterparts in IPv4 and IPv6, RINA addresses in different DIFs are taken from different name spaces

[4]Even if they are simply null

[5]This assumes the attacker also has sufficient computing resources to flood those paths.

V. How could an attacker get the required information to attack a DIF?

In this section we will discuss how an attacker might go about obtaining the information required to attack the various aspects of a DIF as described in the preceeding sections. We will be examing the RINA API calls as well as the Common Distributed Application Protocol (CDAP).

A. RINA API

In order to understand what information may be available to the attacker we will examine the data flow of the RINA API. The RINA API has 5 primitives: `Allocate_Request`, `Allocate_Response`, `Send`, `Receive`, and `De-Allocate`. We will first examine these primitives from the perspective of the servicer and follow up by examining the perspective of the caller. Another way to look at this is to say that for a given N-DIF IPC Process, there are two trust boundaries at the API level. One between the N-DIF IPC Process and the one in the DIF above (N+1)-DIF[6], and another between the N-DIF IPC Process and the one below it (N-1)-DIF[7]. See Figure 1.

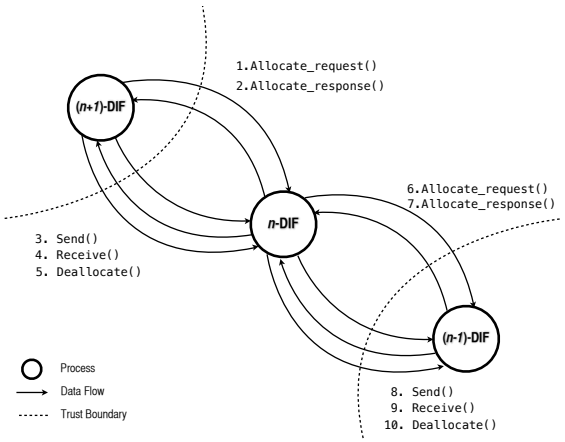

Fig. 1. RINA API Data-flow

Allocate_Request: The `Allocate_Request` API is used by an AP when it wants to initiate an exchange of SDUs with another AP. The call includes the Destination application name to allocate a flow with, the source the application identifier of the application making the request (e.g. itself), the requested quality of service parameter, and the access credentials to use to access the destination AP. In response, the API provides a port-id, similar to a file handle, to be used to refer to the established flow on future calls, and a reason code and message indicating success or failure

Allocate_Response: The `Allocate_Response` API call is used to respond to a request for allocation. Input to the call is as follows. The destination that the AP is responding to. The available QoS that will be used and the port that was provided in the corresponding `Allocate_Request`. The response is simply Reason code / message.

Send: The `Send` API is used to send an SDU to the remote end of the allocated flow. The call includes the port-id that identifies the flow to send the SDU on, and the buffer containing the SDU to send. The response to this is a reason code indicating success or failure.

Receive: The `Receive` API is used to read SDUs sent from the remote end of the flow. The call includes the port-id indicating which flow to read from and a buffer to write the SDU into. The output of this call is the data written to the specified buffer and a reason code indicating success for failure.

De-Allocate: The `Deallocate` API is used to indicate that all resources associated with the port-ids should be de-allocated. (The state associated with the connection will disappear after 2 or 3 delta-ts.) The single parameter passed to this API is the port-id indicating which resources should be de-allocated. In response is a reason code indicating success or failure.

1) (N+1)-DIF → N-DIF: Let us now examine the 5 API calls across the trust boundary between caller and servicer where the caller is a layer above the servicer. We will assume some level of distrust of the (N+1)-DIF.

Data Flow 1: Allocate_Request: In order to service this call, the IPC Process servicing this request will have to make use of information an attacker is interested in. This includes the DIF name, the address of the destination, the DIF's PDU protection policy, and any key material necessary to meet the PDU protection policy. In general it is not difficult to keep this information from the calling AP, since all that is required to be returned is a port-id. The IPC Process should take care not to leak any of the aforementioned information via the reason code (e.g. including the data in overly helpful error messages). For example an IPC process should take care not to offer up any hints as to the DIF namespace when the caller passes an invalid destination AP name. On the flip side, the IPC process should take care to validate the incoming Allocate_Request calls against malformed arguments. For example, the IPC process should not allow a caller to pass a source application identifier other than its actual identifier. The QoS parameters are not much of a threat, and require only basic input validation (e.g. bounds checking, invalid character sequences, etc.). The access credentials will be validated by the destination AP and so only basic local validation is required.

Data Flow 2: Allocate_Response: When this is coming from the DIF above, it should correspond to an earlier request. That means that servicing IPC process should keep track of the source and port-id of Allocate_Requests it has sent to a given AP and only accept Allocate_Response calls with corresponding destination and port-id; all others should simply be discarded. The QoS parameters require only basic input validation. The IPC process should take care not to leak information about the DIF (DIF Name[8], addresses, etc.) via the reason code when responding with an error condition.

Data Flow 3: Send: The send API is only valid for flows that are established, so the N-DIF IPC process should validate

[6]can also be an Application Process

[7]can also be the physical medium

[8]Included for completeness. The DIF Name is easily known, but an attacker must still know it to launch an attack.

that the specified port-id is associated with the calling AP. In addition input validation should be done on the SDU buffer. As with earlier calls, the IPC process should be careful not to divulge any information about the DIF when responding with an error condition.

Data Flow 4: Receive: Like the send API, the receive API is also only valid for established flows. The N-DIF IPC process should validate the specified port-id is associated with the calling AP before filling the specified SDU buffer with data from the flow and validate that the incoming SDU will fit into the indicated buffer. The IPC process should also take care not to divulge any DIF-sensitive data when responding to error conditions.

Data Flow 5: De-Allocate: As we discussed with Send and Receive, the IPC Process should validate that the port-id provided corresponds to a flow that is allocated to the calling AP. Also, like the preceding APIs, the IPC Process should take care not to divulge any DIF-sensitive information to the caller via error messages.

2) N-DIF → (N-1)-DIF: Now we will consider the trust boundary between N-DIF and (N-1)-DIF assuming some level of distrust of (N-1)-DIF.

Data Flow 6: Allocate_Request: When making the call to Allocate_Request, the N-DIF is divulging the destination AP name, its own identifier, the quality of service it thinks the destination can provide, and access credentials. The AP name is not particularly useful to the attacker other than perhaps to track what APs the source wants to connect to. The source application name may be useful to an attacker wishing to impersonate the source. The quality of service parameters may be useful to an attacker since the level of service may indicate what type of bandwidth is available between the source and destination. A high amount of bandwidth may indicate a high profile target. A low amount of bandwidth may indicate an easier target of a DoS attack.

The access credentials are quite useful to an attacker especially if they are unprotected. Either way, some care must be taken to ensure that an attacker can neither learn the cleartext credentials or use them in a replay attack against the destination AP. Some care should be given to the number of retries that are attempted; if the authentication scheme in use relies on a sequence, then an attacker may use failed authentications to attempt to learn enough of a sequence to establish a seed.

The port-id is provided by the lower level DIF and is of little interest. The N-DIF IPC process or AP should take care to properly handle requests coming from a port-id that was already allocated prior to the new request.

Data Flow 7: Allocate_Response: In this case, the IPC process should take care to validate the access credentials that came along with the corresponding Allocate_Request before making any response. Since we are assuming that the (N-1)-DIF is not fully trusted, then we must consider that the lower level DIF could attempt a Man-in-the-Middle attack. The response data should be protected in some way otherwise after passing along the access credentials the (N-1) IPC Process could use the response to insert itself into the flow and send SDUs to the accepting AP when the AP thinks it is talking to the original requester. If the Allocate_Request is not authenticated, then an attacker could learn the available QoS by sending repeated requests with different levels of QoS. The port-id is supplied by the lower level DIF and therefore is of little interest from a security perspective.

Data Flow 8: Send: There are only two pieces of data of interest from the API perspective when calling send: port-id and the SDU buffer. The port-id is supplied by the (N-1)-DIF and is therefore uninteresting from a security perspective in this case. The SDU on the other hand is potentially very valuable to an attacker. Care should be taken to protect the SDU with encryption and integrity checking depending on the level of security required. If encryption is employed, care should be taken to never pass an SDU buffer that has previously contained clear text. If this is unavoidable due to resource limitations, care should be taken to zero out the clear text remaining in the buffer before passing the data via the send API. If SDU protection is not used, the N-DIF process should assume that the contents of any SDUs sent are known to any AP in the DIF through which the SDU passes.

Data Flow 9: Receive: There are three pieces of information to consider for the receive call: the port-id, the buffer reference, and the data that gets placed into the buffer. The port-id is a value that is provided by the lower level DIF and as with earlier calls is not very interesting security-wise. Care should be taken by the N-DIF process that the buffer provided into which data should be written does not contain any clear text from earlier messages. Once the data has been received, validation of the buffer and integrity checks on the decrypted SDU should be done. If either fails, the SDU should be discarded.

Data Flow 10: De-Allocate: The deallocate API may seem uninteresting from a security perspective at first, but if the (N-1)-DIF is not trusted and it is critical to the N-DIF process that the flow actually be deallocated, then the N-DIF process should indicate to the AP on the other side of the flow, within the application protocol, that it will deallocate before making the actual deallocate API call. This will help to ensure that the (N-1)-DIF IPC Process cannot fool the AP at the other end of the flow into continuing the connection[9].

B. CDAP

Thus far in our examination of how an attacker might obtain enough information to launch an attack against a DIF, we have been looking at the boundary between DIFs which occurs on a specific host. We must now turn our attention to data that gets exchanged between hosts[10]. The protocol used by DIF members to coordinate operation of the DIF is called the Common Distributed Application Protocol (CDAP).[5] CDAP has 9 actions: Connect, Release, Create, Delete, Read, Cancel Read, Write, Start, and Stop. Each action has a request and response message for a total of 18 PDUs that can be forwarded among DIF members. Each message contains a subset of a possible 23 fields (See figure 3).

[9]This is likely to only be an issue if SDU protection is not being employed.

[10]The information may also apply to Different APs communicating via an IPC process on the same host.

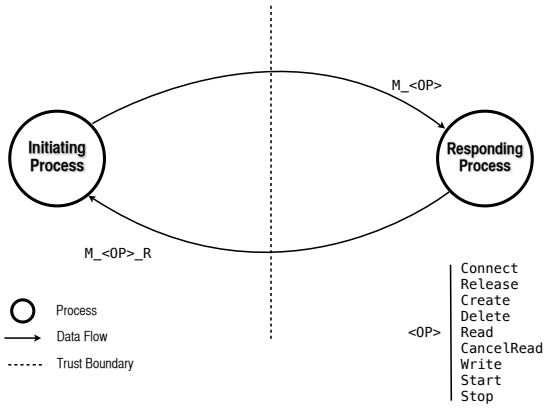

Fig. 2. CDAP Data-flow

The following descriptions are from the CDAP reference specification.[5]

Connect
: Initiate a connection from a source application to a destination application.

Release
: Orderly close of a connection.

Create
: Create an application object

Delete
: Delete a specified application object

Read
: Read the value of a specified application object

Cancel Read
: Cancel a prior read issued using Read for which a value has not been completely returned

Write
: Write a specified value to a specified application object

Start
: Start the operation of a specified application object, used when object has operational and non-operational states.

Stop
: Stop the operation of a specified application object, used when then object has operational and non-operational states

The primary trust boundary exists between the local host and the rest of the world.[11] Let us now consider each of the messages as they cross the boundary from either direction. (See figure 2)

1) World → Local Host: First we will examine the security aspects of each message from the perspective of an IPC process within the trust boundary on the local host. When processing each of the requests discussed below, a local IPC process should do an authorization check on the requester before offering up any data.

Data flow 1: Connect and Connect_R: When a connect message comes in, the servicer should take care to validate the incoming source and destination identifiers. It should also authenticate the sender if the authentication fields are populated and the authentication policy specifies it. When sending a response the IPC process should take care not to leak any information that may allow the attacker to build up knowledge of the DIF's namespace. If the IPC process is handing its authentication directly it should consider using standard best practices for preventing attacks such as not accepting further connection requests after some number of specified failed authentications.

Data flow 2: Release and Release_R: The primary concern when processing this message would be to verify that the release request is actually coming from the DIF process for which the request is being made. The message itself does not contain enough information to verify this other than comparing it against a table of open connections. CDAP will have to rely on lower-level mechanisms such as overall SDU protection or a session identifier established external to CDAP in order to protect against errant or fabricated messages. It is not currently specified by the CDAP reference, but a potential way to guard against fabricated release messages would be to use the invokeID to pair release messages with connect messages.

When responding to a release request, care should be taken not to divulge information about the namespace of the DIF or allow an attacker to map out what connections are active. For example, if a request to release a connection that is not active or was never started, responding with a success status would prevent an attacker from learning what connections are active. This is especially true when the AP or AE names are malformed.

Data flow 3: Create and Create_R: Aside from verifying that the requester is authorized to create the specified object(s), the IPC process servicing this request should take care to validate the scope and filter values to be sure that the caller is not able to elevate their privilege via this mechanism. If an object value is specified in the message, this should also be validated against the class to be sure the value is appropriate. The size of the value should also be validated to fit within the allocated space.

When sending the response for this request care should be taken to limit the amount of information divulged to the caller especially if the object to be created would not be authorized for the given caller.

Data flow 4: Delete and Delete_R: Delete and Delete_R have similar processing concerns to Create and Create_R. Except that since no object values are provided, validation of that field is not required.

Data flow 5: Read and Read_R: Read and Read_R have similar concerns to Delete and Delete_R.

Data flow 6: Cancel Read and Cancel Read_R: Cancel Read and Cancel Read_R have similar processing concerns to Release and Release_R except that in this case there is an additional way to validate the request. The pair of invokeID and object instance ID must match an existing read request in

[11]A second trust boundary is between the local host and members of the DIF. A third trust boundary is between the local host and non-DIF members. A fourth boundary may be considered also if some members of the DIF are trusted more than others, but we will leave these as topics for further study.

progress.

Also like Release_R, care should be taken to limit the amount of information divulged via error messages for errant or fabricated Cancel Read requests.

Data flow 7: Write and Write_R: Write and Write_R have similar processing concerns to Create and Create_R.

Data flow 8: Start and Start_R: Start and Start_R have similar processing concerns to Create and Create_R, although some additional care should be taken to not allow an attacker to determine which objects have operational states.

Data flow 9: Stop and Stop_R: Start and Start_R have similar processing concerns to Start and Start_R.

2) Local Host → World: Now we will turn our attention to each of the messages from the perspective of the local host sending request messages to other DIF processes.

Data flow 10: Connect and Connect_R: The information of interest to an attacker in this message includes the destination AE and destination AP names as well as the names of the source AE and AP. If included, the instance names are also of value. All of these values combined, especially when combined with data from other nodes, may allow an attacker to derive the DIF namespace or at least a portion of it. This would give the attacker an advantage in guessing the address or name of a specific node in the network other than the ones specifically named in the intercepted connect message. Also of interest to an attacker from the Connect message are the authentication credentials.

The CDAP API itself does not seem to provide a way to authenticate the destination AP. This would have to be done externally either by querying an already trusted DIF member or via an AP specific mechanism (e.g. something similar to 2-way SSL in the TCP/IP architecture).

To protect the information required by this pair of messages, SDU protection that includes a confidentiality component (e.g. encryption) should be employed.

Data flow 11: Release and Release_R: From the sender's perspective there is not much that can be leaked via a Release message. Even if a Release_R is never received, all future traffic from this flow can be ignored. Theoretically if there is a rogue IPC process between the sender and receiver, it could potentially (in the absence of proper SDU protection) take over a connection if say it is on the receiving end of publish/subscribe conversation and the sender doesn't first issue an unsubscribe.

Data flow 12: Create and Create_R: The Create and Create_R messages contain valuable information. In order to use the API as intended, the sender must provide the object class, an object name, and an object instance identifier. It can optionally provide an initial value for the object. If PDU protection is not being employed then an attacker may have direct access to this data. In addition, the filter and scope fields may reveal even further information about the structure of the object namespace. Unsolicited Create_R messages should be flagged as potentially critical events.

Data flow 13: Delete and Delete_R: Delete and Delete_R have similar concerns to Create and Create_R except that the values of the objects being deleted are not divulged. Unsolicited Delete_R messages should be flagged as potentially critical events.

Data flow 14: Read and Read_R: Read and Read_R have similar concerns to Create and Create_R.

Data flow 15: Cancel Read and Cancel Read_R: Cancel Read seems to be the least risky message that can be sent in CDAP. The message itself contains no critical fields. The only thing to be cautious of is divulging too much information via a reason code or message.

Data flow 16: Write and Write_R: Write and Write_R have almost identical concerns to Create and Create_R.

Data flow 17: Start and Start_R: Start and Start_R messages have similar concerns to Create and Create_R but in addition to naming the object and potentially its value, sending a Start message indicates to an eavesdropper that the object in question has at least one operational state.

Data flow 18: Stop and Stop_R: Stop and Stop_R have similar concerns to Start and Start_R.

C. Other methods of obtaining required information

Now that we have explored the RINA API and CDAP messages, we will examine briefly some additional potential methods for obtaining enough data to attack a DIF. Most of these are areas for further study. In the absence of a confidentiality component included in the PDU protection policy, it seems feasible that any node in a lower level DIF shared with a target may eavesdrop on the PDUs it sends. If the target also does not employ authentication against the APs it connects with, then this seems to open up that target to man in the middle or masquerading attacks. If an attacker is able to get access to a RIB out of band or gain access to it via the API, they will have a treasure trove of data with which to launch targeted attacks against any DIF member who's data is stored even partially in the compromised data store.

VI. Attack Mitigations

In this section we will discuss mitigations to the attacks and data leakage explored in previous sections. There are four major mitigations that can be employed to strengthen a RINA network: 2-way authentication, PDU encryption, PDU integrity protection, and strong auditing. This seems uninteresting at first, but upon further reflection indicates that the standard security tools: Authentication, Authorization, Confidentiality, Integrity Protection, and Auditing are sufficient to secure a RINA network.

VII. So, is a DIF a Securable Container?

Now it is time to pull everything together and revisit our starting question: *Is a DIF a securable container?* In short, yes. While we recognize that this analysis is limited by its reliance upon an implementation constructed using secure coding practices, the architecture itself does not introduce new security flaws. When proper security tools are used, a DIF is a "structure used to hold or transport something that can be made to be not subject to threat." As we have shown, given a properly implemented transport protocol, when SDU protection includes a confidentiality component, all critical data that is otherwise available for capture becomes opaque. If

in addition to this, communicating APs mutually authenticate, attackers have little opportunity to obtain enough information to launch successful attacks against specific, random, or collective DIF targets. A RINA DIF is a securable container as we have defined it.

Some areas for further study include: a threat analysis of EFCP the proposed suite of transport protocols for RINA, a threat analysis of additional trust boundaries within a DIF such as when a subset of DIF members are trusted more than others[12], attacks on DIF traffic as it passes through lower level DIFs, and other out-of-band methods for obtaining data from the RIB such as reading the data directly from long term storage.

REFERENCES

[1] John Day *Patterns in Network Architecture: A Return to Fundamentals*. Upper Saddle River, N.J. : Pearson Education, c2008.

[2] J. Day, I. Matta, and K. Mattar, *"Networking is IPC": A Guiding Principal to a Better Internet*. in Proccedings of Rearch'08 - Re-Architecting the Internet Madrid, SPAIN: Co-located with ACM CoNEXT 2008, December 2008.

[3] G. Boddapati, J. Day, I. Matta, and L. Chitkushev *Assessing the Security of a Clean-Slate Internet Architecture* Boston, Massachusetts, May 2010.

[4] New Oxford American Dictionary 2nd edition 2005 by Oxford University Press, Inc.

[5] S. Bunch *CDAP - Common Distributed Application Protocol Reference*. Unpublished, December 2010.

[6] J. Day *Patterns in Network Architecture: Recursive IPC Network Architecture: The Reference Model: Basic Concepts and Distributed Applications*. Unpublished, 2009.

[7] J. Day *Patterns in Network Architecture: Recursive IPC Network Architecture: The Reference Model: Distributed InterProcess Communication*. Unpublished, 2009.

[8] J. Day *Patterns in Network Architecture: Recursive IPC Network Architecture: The Reference Model: Operations*. Unpublished, 2009.

[9] J. Day *Patterns in Network Architecture: Recursive IPC Network Architecture: The Reference Model: Interesting Configurations*. Unpublished, 2010.

[10] Dieter Gollmann *Computer Security, 2nd Edition*. San Francisco, CA : John Wiley & Sons 2006.

[12] Such as in a service provider peering DIF.

VIII. Additional Figures

GPB name	M_CONNECT	M_CONNECT_R	M_RELEASE	M_RELEASE_R	M_CREATE	M_CREATE_R	M_DELETE	M_DELETE_R	M_READ	M_READ_R	M_CANCELREAD	M_CANCELREAD_R	M_WRITE	M_WRITE_R	M_START	M_START_R	M_STOP	M_STOP_R
absSyntax	M	M																
authMech	A	A																
authValue	A	A																
destAEInst	A	A																
destAEName	M	V																
destApInst	A	A																
destApName	M	V																
filter					A	V	A	V	A	V			A	V	A	V	A	V
flags	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
invokeID	M	=	A	=	A	=	A	=	A	=	=	=	A	=	A	=	A	=
objClass					A	V	A	V	A	V			A	V	M	V	M	V
objInst					M	V	M	V	M	V					M		M	
objName					M	V	M	V	M	V					M		M	
objValue					A	A				C			M	A	A	A	A	V
opCode	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
resultReason		C		C		C		C		C	C	C		C		C		C
result		M		M		M		M		M	M	M		M		M		M
scope					A		A		A				A		A		A	
srcAEInst	A	A																
srcAEName	M	V																
srcApInst	A	A																
srcApName	M	V																
version	M	M	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

Fig. 3. CDAP Message Detail [5]