Resource Public Key Infrastructure Denial of Service Assessment

Raytheon BBN Technologies
Andrew Chi, Steve Kent, Mark Reynolds
July 22, 2011

Contents

Introduction

The Resource PKI

The Resource Public Key Infrastructure (RPKI) is the public key infrastructure for tracking the allocation/assignment of Internet number resources. The RPKI supports improved security of Internet routing by securely authorizing the hierarchical mapping between Autonomous System (AS) numbers and IP address space. Using the RPKI, a holder of IP address space can issue a resource certificate chain which eventually terminates in a signed Route Origination Authorization (ROA). This ROA verifiably authorizes one or more ASes to originate routes to the holder's address space. The current description of the proposed RPKI architecture can be found in [draft-ietf-sidr-arch].

The RPKI consists of a large set of digitally signed objects attesting to resource number holdings (address space and Autonomous System numbers), and a distributed repository system that acts as the clearinghouse for publishing those signed objects. Resource holders such as IANA, the RIRs, NIRs, and LIRs/ISPs, will hold CA certificates and issue subordinate CA certificates, end-entity (EE) certificates, and signed objects such as ROAs and manifests. Each resource holder uploads its signed products to a repository server administered by the resource holder or a repository operator, e.g., a regional or national registry.

Relying parties (RPs) consume the information in the repository system. A typical example of a relying party is an ISP that participates in BGP routing, and therefore needs to determine which route advertisements to accept from its neighbors. Each RP synchronizes its local copy of the repository with all external repositories, then validates the AS-to-IP mappings via object signatures, and finally uses the validated information to configure its routers. Raytheon BBN Technologies has developed prototype RP software that retrieves RPKI objects, validates certificates, ROAs, manifests, and compound trust anchors and produces router-friendly output listing valid AS-to-IP prefix mappings. The Raytheon BBN RP software is the subject of this assessment.

Security Considerations for Relying Parties

Although each digitally signed object in the RPKI is cryptographically protected, an adversary is by no means limited to direct attacks on cryptography. Under normal operation, both the distributed repository system and the relying parties must process large amounts of data from various sources, all of which is initially untrusted. In addition, the current draft for the repository system [draft-ietf-sidr-repos-struct-04] explicitly states that the repositories themselves will not be "protected" structures, and thus retrieval operations by RPs are vulnerable to various forms of "man-in-the-middle" attacks.

An adversary who wishes to nullify the benefits of the RPKI could do so by conducting a denial-of-service attack against the RPs, and in particular, the RP software. The architecture requires RPs to acquire all RPKI objects, so a single serious flaw in the RP software could allow a single malicious object at a single repository to degrade RPKI operation on all RP machines running that software. Thus, a properly functioning RPKI requires deployment of secure RP software

that can produce valid, up-to-date AS-prefix mappings, even in the presence of motivated and capable adversaries.

In this assessment of the Raytheon BBN RP software, we develop a threat model for adversaries whose overall goal is denial-of-service, we enumerate the vulnerabilities we have identified in the current release of the RP software, and we propose mitigations.

BBN Relying Party Software

The BBN Relying Party software is a suite of programs that retrieve data from the distributed repository system and create router-friendly prefix origin output. For efficiency reasons, the programs are run periodically in the background so that as new signed objects arrive, the ROA information is incrementally updated. Figure 1 shows the major components, which are described below.

RP Software Architecture

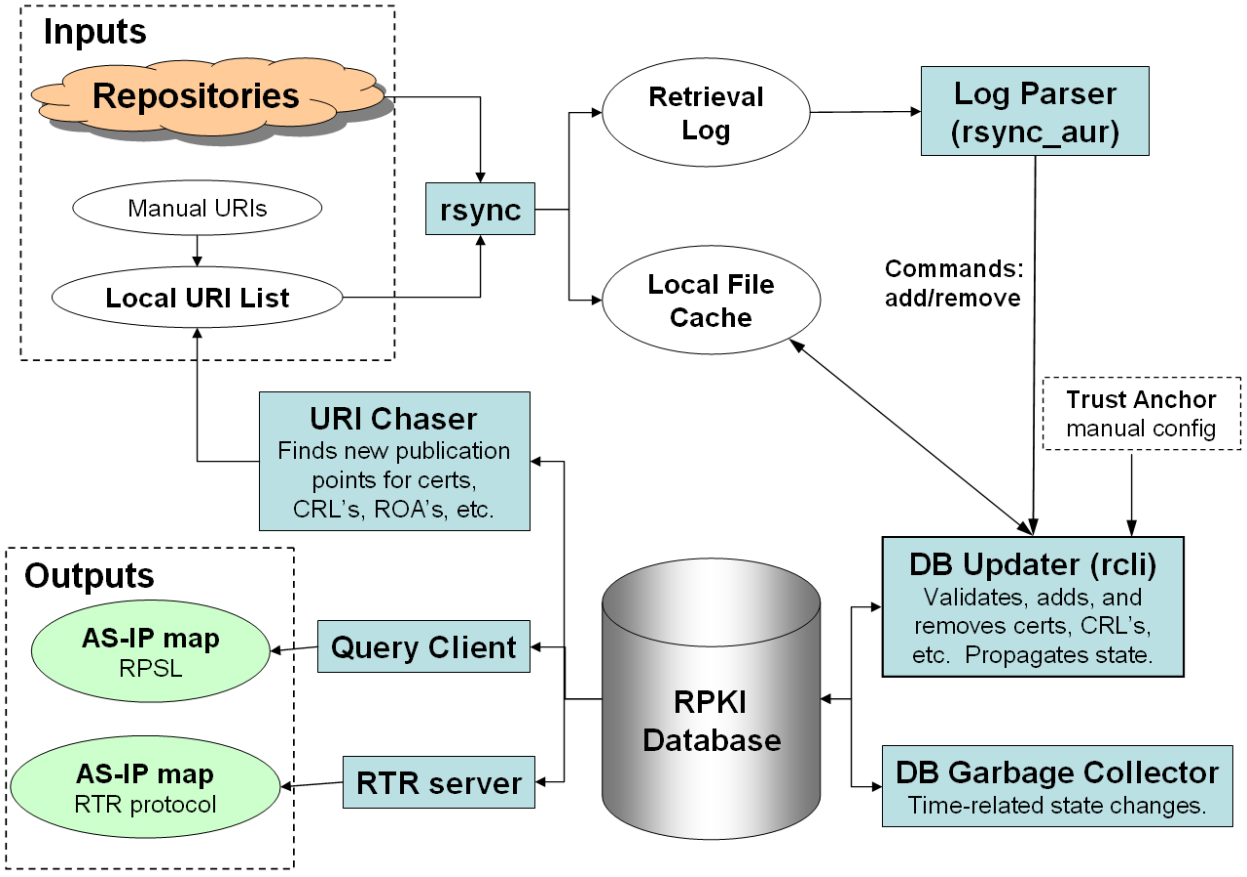

Figure 1: RP Software Architecture. Arrows represent the direction of data flow and logical read/write relationships. White ovals are locally-stored files. Blue boxes are executable components.

- **Rsync.** Rsync (a file system synchronization protocol) serves as the main input interface of the Raytheon BBN RP software. Since most signed objects appear and disappear slowly over time, using rsync efficiently retrieves the differences only. An rsync wrapper script reads the local URI list (see below) from a file, and for each URI, invokes rsync to synchronize part of the local file cache (local repository) against that URI. Though rsync can be used for bidirectional synchronization, in the RP software, rsync only reads from the remote repository publication points. Rsync is configured to generate a retrieval log of all files which were added, updated, or removed.
- **Local URI List.** The Local URI List is an ordered list of currently active repository publication points. Except as noted elsewhere, these are rsync URIs. This list is initially (manually) configured with a set of repositories such as those managed by IANA and the RIRs, and subsequently augmented by the URI Chaser as each retrieved signed object designates its AIA, SIA, and CRLDP. It also may contain local file system URIs if local trust anchor management mechanisms are employed [draft-reynolds-rpki-ltamgmt-00].
- **Local File Cache**. As files are retrieved from the distributed repository system, they are stored in a local repository, a directory tree that mirrors the (global) repository structure. The DNS name of each publication point serves as the highest-level directory name for data from that server. For example, a certificate on an APNIC server might be cached at *$CACHEDIR*/apnic.mirin.apnic.net/path/to/foo.cer.
- **Retrieval Log.** As objects are retrieved from remote repositories, rsync writes a retrieval log, creating one line per file that is added, updated, or removed. This retrieval log is read by the Log Parser, and then cleared.
- **Log Parser (rsync_aur).** Retrieval operations must be translated and imported into the database. The first step of this process is performed by the Log Parser. The Log Parser reads the retrieval log and interprets each file retrieval operation as an add, update, or remove, and sends a message of this action across a TCP connection to the DB Loader/Updater.
- **Relational Database (MySQL)**. The MySQL database is used to enable rapid location of objects, and it provides centralized data access for most components of the RP software. The database has five main tables, one for each type of object: certificates, certificate revocation lists (CRLs), route origination authorizations (ROAs), manifests, and compound trust anchor data structures. If an object is determined to be invalid, that object will either not be loaded into the database, or be deleted from the database. Moreover, for simplicity and efficiency, not all fields of each type of object are represented by the database. The only fields included are those required for searching/identifying objects. Seldom-used information can be recovered from the corresponding file in the local repository if needed.
- **DB Updater (rcli).** The DB Updater is the main validation component of the RP software. It runs continuously in the background and has read/write access to the database. The DB Updater listens on a TCP port for notifications of added, updated, or removed signed objects, validates them, and inserts them into the MySQL database, propagating any state that must change due to the new/updated objects. The DB Updater can also be specially invoked out-of-band for the purpose of configuring a trust anchor. A detailed description of DB Updater algorithms is given in the relevant section of the vulnerability analysis.

- **DB Garbage Collector (gc).** The DB Garbage Collector runs periodically and handles all time-related state changes. It has read/write access to the database. For example, a certificate that expires or enters its validity period will cause state changes not only for itself but potentially all of its children; a CRL or manifest that becomes stale will call into question the validity of its sibling certificates. A detailed description of the garbage collection algorithms is given in the relevant section of the vulnerability analysis.
- **URI Chaser.** The URI Chaser searches the database for new publication points. RPKI certificates contain the fields Authority Information Access (AIA), Subject Information Access (SIA), and CRL Distribution Point (CRLDP) extensions. The URI Chaser searches the database for all AIA, SIA, and CRLDP, and eliminates duplicate and subsumed URIs (subdirectories of existing URIs). If there are new URIs, the Chaser kicks off a cycle of rsync/Log Parser/DB Updater.
- **Query Client.** The Query Client is a command line utility that produces RPSL output. It is a read-only consumer of database information, and performs ROA validation in order to determine the set of currently valid route origins. The RPSL format is the current product generated for an RP, chosen because it can be consumed by extant ISP tools used for route filter generation.
- **RTR Server.** The RTR Server implements the proposed RPKI/Router Protocol, a mechanism for delivering prefix/origin data to routers in an AS from a server for that AS, protected via SSH [draft-ymbk-rpki-rtr-protocol-05]. The RTR Server functions as a read-only consumer of RPKI database information. The RTR Server periodically creates a copy of the valid ROA information in auxiliary database tables, and handles client (router) requests for prefix origin data by reading those tables.

More detailed descriptions of the BBN Relying Party software can be found in [12] and [14].

Threat Model

The basis of this vulnerability analysis is a model of the potential *threats*, i.e. motivated, capable adversaries. Since this is a denial-of-service (DoS) analysis, the threats of interest all have the common motive of denying or degrading the services provided by the relying party software: that is, preventing the RP software from delivering valid, up-to-date associations between AS numbers and IP prefixes. However, different threats may possess different capabilities, depending on their level of access to the RPKI.

Adversaries by Capability Class

The scope of the threat model is the set of adversaries whose computational capabilities are insufficient to break the cryptographic primitives, and who have not already compromised a significant fraction of the RPKI hierarchy. Specifically, we assume that a direct attack against the cryptographic algorithms used in the RPKI is infeasible, even for nation-state adversaries, and that the top level CAs, such as IANA and the RIRs, will adequately protect their private keys. We define *insider* to be an entity that is authorized to participate in the RPKI, and we define *outsider* to be any other entity. The threats of interest consist of three capability classes:

- **Outsider:** An outsider (e.g., hacker) holds no resources and thus possesses no valid RPKI certificates, i.e. certificates that have a valid certificate path beginning at a widely-accepted trust anchor. However, an outsider is capable of signing arbitrary objects, and

depending on the authentication model of the repository system, he can (1) upload invalid objects to a repository, or (2) impersonate a repository using some form of man-in-the-middle attack, or both. Note that if the repository authentication model fails to prevent (2), then an outsider has nearly the same capability as a Repository Manager.

- **Insider – Certification Authority:** In the RPKI, all resource holder participants must be able to issue certificates and other signed objects, and thus all are CAs. (The policies of the RIRs do not authorize all resource holders to sub-allocate resources, so not all need to be able to issue subordinate CA certificates. However the RPKI does not attempt to enforce this policy via technical means.) Each CA holds current and previously valid certificates, and can upload valid certificates and signed objects to its publication point in the global repository. As with outsiders, depending on the repository authentication model, CA also may be able to upload to other publication points.
- **Insider – Repository Manager:** A repository manager (RM) controls a repository server and can modify server behavior. In particular, an RM can add or delete arbitrary files in the repository publication points on that server. RMs themselves are unable to create valid RPKI certificates (if they are not also CAs). However many CAs are expected to be RMs, and a malicious CA might work in concert with a malicious RM.

Note that any CA that fails to properly protect its private keys is equivalent (from a threat perspective) to a malicious CA. Also, it is important to distinguish two types of insider attacks. This first is self-sabotage of the signed objects for which the insider is considered authoritative: such "shooting oneself in the foot" cannot be addressed by technical measures available to RPs. The second is an insider attack that affects the availability of objects owned by *other* CAs or repositories: this is a far more dangerous behavior and an appropriate threat mitigation goal.

Adversary Objectives

In the context of the BBN RP software, an adversary with the top-level goal of DoS may choose any of the following major objectives, each of which could degrade or deny service. In an attack tree, these nodes would be the direct children of the root node.

- Exhaust CPU
- Exhaust bandwidth
- Exhaust memory
- Exhaust disk storage
- Inhibit timely output
- Terminate RP software
- Cause high error rates
- Unsuccessful certificate/object validation
- Unauthorized access within RP software and data
- Unauthorized system-level access

Note that the final three objectives could achieve more serious consequences than simple DoS, since they can result in the RP software delivering maliciously-crafted output, not simply out-of-date output or no output.

In general, DoS can occur any time it costs an adversary less to send/create data than it costs the recipient to process it. This is commonly seen in client-server DoS attacks. In the RPKI

architecture, the number of affected parties is multiplied because all relying parties must acquire all data from all repositories.

It is desirable for RP software to adhere to the principle of least privilege (confinement). That is, certificates and other objects should affect only their subtree of the RPKI. A malicious CA or RM can always deny availability of resources that it was authorized to allocate or publish—this is unavoidable. However, in a properly designed system, a malicious entity should be able to affect *only* its own resources or resources subordinate to it in the allocation hierarchy. Any aspect of the RP software that fails to adhere to the principle of least privilege can be considered a DoS vulnerability.

Relevant and Irrelevant Attack Vectors

To achieve his objectives, a capable adversary will employ methods tailored to the specific components and data flows of the target system. We give an overview of the potential targets and attack vectors that we consider relevant to the BBN RP software. We also state which attack vectors we intentionally dismiss as irrelevant or out-of-scope.

We consider the following attack vectors relevant to DoS threats. These categories are not intended to be disjoint; instead they provide an overview of potential approaches.

- **Executable components.** All executable components in the RP software are potential targets. The RP executable components comprise rsync, Log Parser, DB Updater, DB Garbage Collector, MySQL database, URI Chaser, Query Client, RTR Server, and any scripts that "wrap" these executables.
- **RPKI Objects.** The RPKI distributed repository system is responsible for publishing four types of signed objects: route origination authorizations (ROAs), X.509 Certificates with RFC 3779 extensions (certificates), X.509 Certificate Revocation Lists (CRLs), and manifests. A fifth type of signed object is distributed out-of-band but is also handled by the RP software: the Compound Trust Anchor, see [draft-ietf-sidr-ta-04]. The four types of signed objects published by repositories can all be constructed maliciously and delivered by an adversary, and thus are attack vectors of interest.
- **Data Storage.** The Local Repository (File Cache) provides an on-disk mirror of the distributed repository structure. In addition to the usual security considerations of storing and manipulating untrusted data, it is important for the RP software to properly isolate the data based on its source. Signed objects from one repository should not be able to affect signed objects from another repository, except through the cryptographically authorized validation and revocation processes defined in the RPKI.
- **Input/Output.** As with any network-capable system, all network inputs must initially be considered untrusted. Unsanitized data can result in attacks on input parsers (e.g. buffer overflow), or on multi-layered syntax (e.g., SQL injection). In general, any network property such as bandwidth or blocking behavior also must be considered.
- **Supporting Libraries.** The BBN RP software relies on several open source libraries/packages: rsync, OpenSSL, Cryptlib, and MySQL/ODBC. We assume that the adversaries can exploit any publicly known vulnerability in this software.
- **Server Configuration – External Security.** The server running the RP software needs to be secured against external attack. In particular, since the RP software consists of

several executable components that transfer data among themselves, it is important that the inter-process communication channels be secured against external hijacking.

We intentionally dismiss the following attack vectors as irrelevant or out-of-scope.

- **Cryptography.** We assume that a direct attack on cryptography is infeasible, even for nation-state adversaries. On the other hand, we do not assume that all resource holders will adequately protect their private keys. As stated above, a CA that does not protect its private keys is equivalent to a malicious CA.
- **Confidentiality of RP data.** The RP source code will be made freely available. In addition, no private keys are required for the successful operation of the RP software. Thus, except for administrative credentials, all data associated with the RP software can be considered public.
- **Trust Anchor Configuration.** Trust anchor material is assumed to be configured through an out-of-band, trusted process. In particular, the compound TA object is dismissed as a potential attack vector.
- **Self-Sabotage.** A malicious insider could simply delete or revoke any certificates for which it is considered authoritative. This is unavoidable—in a PKI, a malicious insider can always sabotage its own subtree. We are instead interested in the much more dangerous scenario where a malicious entity uploads files that confuse the RP software in a way that undermines the availability of signed objects from entities at other (not subordinate) points in the allocation and publication hierarchy.
- **Server Configuration – Internal Security.** We assume that the RP software will run on a dedicated server. Therefore, the other users of the server are not considered threats, and file access controls need not be as strict as they would be on a typical multi-user system. However, the server still needs to be properly secured against external threats.

Vulnerabilities and Mitigations

The following vulnerability analysis is organized by executable component of the BBN RP software. The majority of the signed-object processing occurs inside the DB Updater and Garbage Collector, so that section is subdivided by object type (or combination of types). For each component, we describe the vulnerabilities, give example attacks, and propose mitigations. A few reported vulnerabilities have already been mitigated and do not affect BBN's RP implementation; these vulnerabilities are nevertheless mentioned because they are potentially applicable to any RP implementation.

Definitions

In the tables below, we assign to each vulnerability a severity rating using the following terms:

- High: Compromise results in complete denial of service to the RP software's consumers.
- Low: Compromise results in degraded performance, such as higher latency or somewhat delayed information.

In addition, we assign to each vulnerability a mitigation difficulty, using the following terms:

- Easy: Mitigation is isolated to one component and does not require redesign.
- Medium: Mitigation is well understood but may involve integration of multiple components, or requires redesign.

- Hard: The mitigation requires changes to the draft standard, the mitigation involves some fundamental redesign, or there currently exists no clear mitigation.

We also identify the minimum adversary class necessary to exploit each vulnerability. The insider RM and insider CA threats have non-overlapping capabilities (and are thus not comparable), but both have strictly greater capabilities than an outsider.

Vulnerabilities: Rsync

Rsync and its wrapper form the input interface of the RPKI software: this component is responsible for acquiring all repository data via the Internet. Therefore, of all RP software components, rsync and its wrapper are the most directly exposed to untrusted external data. The following table summarizes the rsync-related vulnerabilities and potential mitigations, some of which overlap or should be combined. Note that if the repositories remain unauthenticated as currently proposed, all of these vulnerabilities become exploitable by an outsider (hacker) conducting a man-in-the-middle attack as a stepping stone.

Minimum Adversary	Vulnerability	Severity	Mitigation Difficulty
Outsider	Known rsync vulnerabilities	High	Easy
Outsider	Unauthenticated repositories	High	Hard
Insider RM	Blocking, sequential traversal of URI list	High	Medium
Insider RM	Rsync wrapper: arbitrary code execution	High	Easy/Medium
Insider RM	Unbounded URI list length	High	Medium/Hard
Insider RM	Unbounded URI chase depth	Low	Easy

Known rsync vulnerabilities. Since rsync is directly exposed to the internet, it can be attacked through all well-known vulnerabilities, such as CVE-2008-1720 (arbitrary code execution), and CVE-2007-6199 (remote access to restricted files). **Mitigation:** Keep rsync up-to-date with the latest patches and system configuration recommendations. Note that fixes to BBN RP software will be necessary if the rsync log format or command line options change between updates. But in principle, the mitigation is straightforward.

Unauthenticated repositories. At the time of writing, a repository is neither required to prove its identity to the RP, nor to establish an integrity-checked connection. This permits a large number of man-in-the-middle attacks. For example, an outsider can impersonate the repository by DNS spoofing, hijacking an existing connection, or rerouting rsync traffic through an adversary node that can modify data in-transit. In particular, if an outsider can impersonate a repository, he can cause the RP's rsync module to perform add, update, or remove operations on any locally cached object for that repository. Fundamentally, this vulnerability is the result of two properties:

- Repositories do not authenticate themselves to the RPs.
- For efficiency reasons, the initial synchronization of repository with local file cache is not cryptographically validated. In particular, an "add" is not validated until the data is delivered to the DB Updater, and a "delete" is currently not validated at all. (An "update" is simply a "delete" followed by an "add".)

Fixing either of the two properties would solve this problem. However, the second property may be difficult to fix, given that validation may require other objects which have not yet been retrieved. Requiring validation for "delete" could lend itself a different denial-of-service attack based on disk exhaustion. **Mitigation:** The repositories should authenticate themselves to the RPs. The authentication method may not need to be as strong as a full PKI, because the goal is simply to isolate repositories from each other. One option that is potentially easy to implement using the existing rsync, is to use SSH to communicate with the repository server. The first connection to each repository must be trusted ("leap of faith"), but all subsequent connections will verify that the server's public key has not changed. Mitigation of this problem requires a change to the repository specification and all corresponding implementations. In addition, all RPKI repositories should be advised to use DNSSEC-protected records, in order to limit opportunities for DNS-based attacks.

Blocking, sequential traversal of URI list. The rsync wrapper traverses the Local URI List sequentially, and for each URI, invokes rsync and the Log Parser. It blocks on these two operations until they complete, and only then does it move on to the next URI. A malicious RM can purposefully set transmission rates to be extremely low, thereby stalling the RP software. A related attack is to publish a large amount of data (say 1 TB) at a publication point. Since the BBN RP software runs rsync serially on the URI list, it will be paralyzed by the malicious repository. **Mitigation:** In order to handle potentially malicious repositories, the synchronization process should be redesigned to access multiple URIs in parallel, so that a single slow (or inaccessible) repository does not impede access to others. In addition, any transfer exceeding a configured time limit or data quota should be terminated and held for manual approval. The redesign will involve creating locks to avoid race conditions and ensure mutual exclusion when necessary. Note that any mutex mechanism must be resilient against malicious repositories attempting to hold a lock indefinitely or otherwise starve out legitimate repositories.

Rsync wrapper: arbitrary code execution. The rsync wrapper is a script that takes a list of publication points, and for each URI, invokes rsync followed by the Log Parser. This script is a proof of concept, and currently does not sanitize its inputs. Therefore, an insider RM could upload an unverified object with an AIA containing shell metacharacters that could be used to execute arbitrary commands (for example, "`rsync://foo.com/; rm -rf *`"). **Mitigation:** Modify the rsync wrapper or rewrite it in a language such as C or Perl, and allow only legitimate input characters to be handed to the shell or other programs. The filter should simultaneously prevent malicious input while permitting all valid inputs.

Unbounded URI list length. The management of the Local URI List currently permits unbounded growth, except for small reductions for eliminating subsumed URIs (i.e., subdirectories). In addition, the Local URI List is ordered: the order determines the sequence of synchronization attempts. An insider RM is capable of creating a large number of certificates with arbitrary AIA, SIA, and CRLDP fields, and could exploit either the unbounded growth of the URI List or insert a large number of malicious URIs preceding desirable URIs. Either of these could result in DoS. This attack is also discussed in the Chaser section. **Mitigation:** Fixing this vulnerability requires modification of the chaser, the rsync wrapper, and the Local URI List. Each entry in the URI List should have an associated state and priority: NEW, ACTIVE, HOLD, DISABLED, ALWAYS. The URI Chaser should initially assign a URI the

"NEW" state with least priority. The rsync wrapper would walk through the URI list by priority, and assign the state ACTIVE after a successful synchronization has occurred. If a URI cannot be reached or exceeds its time or storage quota, the URI should be assigned the state HOLD. The operator should be notified of any URIs on HOLD, which he can manually reassign to the states DISABLED or ALWAYS, depending on the RP operator's assessment of the validity of the URI. The operator should also be able to reassign the priority of any URI. Note that in order to ensure robust behavior of the RP software even in the absence of an operator, entries on HOLD should be removed after a specified period of time, such as a week.

Unbounded URI chase depth. When a new URI appears as the AIA, SIA, or CRLDP field of a certificate, the URI Chaser appends it to the Local URI List, and invokes a cycle of the RP software. Currently, there is no limit to the chase depth, so a malicious RM could post an extremely long chain of signed objects such that the SIA of object *n* points to object *n+1*. Upon receiving the first object in the chain, the RP software will process one object per rsync/update cycle for an arbitrarily long period of time, until the end of the object chain. While this attack does not directly prevent other objects from arriving and being processed, it can consume significant resources. **Mitigation:** For each URI in the Local URI List, record the chase depth. Any new URI that exceeds the chase depth should be put on HOLD for manual approval. Since the depth of the global RPKI tree will be a relatively small number (< 20), it is not unreasonable for RPs to adopt such limits, and the SIDR WG standards might recommend such.

Vulnerabilities: Rsync Log Parser

The role of the rsync Log Parser is to notify the DB Updater of any file changes that have occurred during the last rsync invocation. To do this, the Log Parser reads each entry in the rsync log and sends the appropriate "add", "update", or "remove" command to the DB Updater via TCP. Although the Log Parser is not exposed to untrusted contents of downloaded files, it is still exposed to the untrusted filenames. Recall that if repositories are unauthenticated, then outsiders are just as capable as repository managers, and can inject arbitrary filenames. The following vulnerabilities apply to the Log Parser.

Minimum Adversary	Vulnerability	Severity	Mitigation Difficulty
Insider RM	Standard vulnerabilities	High	Easy
Insider RM	Repository confinement failure	High	Medium
Insider RM	Rsync log format ambiguity	High	Medium
Insider RM	Unauthenticated deletion	Low	Hard

Standard vulnerabilities. Since the rsync Log Parser is exposed to untrusted data through filenames, it is potentially vulnerable to the usual buffer overflow attacks, etc. **Mitigation:** In 2008, the Raytheon BBN RP software underwent an automated analysis, followed by manual inspection for these types of standard vulnerabilities. This analysis needs to be re-run before deployment, in order to cover any new code that has been introduced.

Repository confinement failure. Repository confinement is the general property that each repository, whether malicious or not, can only affect the local mirror data for itself and not other repositories. The only allowed exceptions to repository confinement are the cryptographically

signed actions among objects in the RPKI, such as a certificate revocation which in turn revokes subordinate certificates published at a different repository. If an adversary can find a way to cause repository confinement failure, then he can modify the data from another repository, in effect impersonating that repository. Two specific examples relevant to the Log Parser are given below. **Mitigation:** Whenever possible, strict access controls should be placed on file modifications. For example, a "chroot jail" can limit the ability of the Log Parser to write to the wrong directory of the local cache. When this is not possible, such as in the case of paths stored in the MySQL database, then it is necessary to write input scrubbers that prohibit magic characters and escape strings that would break the confinement property.

Rsync log format ambiguity. The rsync log consists of three essential operations: additions, updates, and removals. Their log entries are formatted as follows:

- `>f+++++++ name_of_added_file.ext`
- `>f.st.... name_of_updated_file.ext`
- `*deleting name_of_removed_file.ext`

Since the log entry format uses control markers that are potentially valid characters in filenames, ambiguity can result. The filename "`foo.cer\n>f+++++++ bar.cer`" contains a control marker as a substring, and would be interpreted as two files. Now, suppose the adversary is a malicious RM with the goal of violating the confinement property above. By naming a junk file appropriately and publishing it under badguy.com, the RM could potentially cause deletion of a certificate in the RP's local cache corresponding to goodguy.com. **Mitigation:** Since the rsync log possesses inherent ambiguity, the mitigation must precede normal log parsing. The solution is to scan the relevant directory to ensure that all files conform to the conventions for naming objects defined by [draft-ietf-sidr-repos-struct-04]. If any non-conforming filenames are found, alert the operator and skip the processing of the current repository's data.

Unauthenticated deletion. When a new object appears at a publication point, the Raytheon BBN RP software provides an efficient but unauthenticated transfer from remote repository to local cache, followed by a slower, authenticated addition to the RPKI DB. This process can be seen in the following diagram:

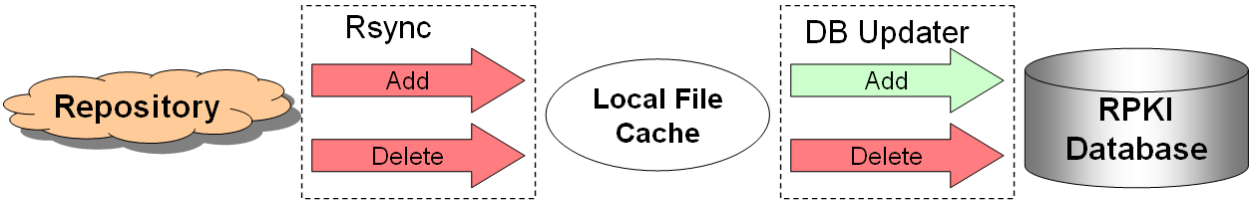

Figure 2: RP Software Unauthenticated Deletion. Arrows represent the direction of data flow. The green arrow is cryptographically verified through the RPKI; the red arrows are not.

It is important to note here that only the final "add" to the RPKI DB is currently authenticated. The left "add/delete" is unauthenticated since it is simply an rsync with an unauthenticated repository. The right-hand "add" is strongly authenticated via the RPKI and its public key signature scheme—but some of this validation may be deferred because not all parent certificates may be available at the time of file acquisition. Note also that right-hand "remove" remains

unauthenticated: this is a known deficiency in the current RP software. **Mitigation:** The primary mitigation is to strongly authenticate deletions for each directory by checking them against the current manifest (for that directory) before deleting the corresponding entries in the RPKI database. Ideally, the second mitigation would be that repositories also authenticate themselves to the RPs, perhaps using a weaker form of identification. The second mitigation would require agreement of the IETF working group, followed by implementation changes.

Vulnerabilities: URI Chaser

The URI Chaser provides an automated way to find new publication points: it periodically searches the RPKI database for all AIA, SIA, and CRLDP fields, filters any subsumed URIs (e.g., file /a/b/c would be subsumed by directory /a/b/), and recreates the Local URI List. Since an object cannot be validated until its parent certificate (pointed to by the AIA extension) is fetched, these URI fields should be considered completely untrusted. As before, if repositories remain unauthenticated, an outsider possesses the same capabilities as a repository manager.

Minimum Adversary	Vulnerability	Severity	Mitigation Difficulty
Insider RM	Unbounded URI list length	High	Medium/Hard
Insider RM	Unbounded URI chase depth	Low	Easy
Insider RM	URI data structure inefficiency	Low	Easy
Insider RM	SQL Injection	High	None
Insider RM	Repository DoS	Low	Hard
Insider RM	Repeated fetching	Low	Medium/Hard

Unbounded URI list length. This vulnerability was already discussed in the rsync section, but is mentioned here again because the mitigation requires bookkeeping changes in the URI Chaser.

Unbounded URI chase depth. This vulnerability was already discussed in the rsync section, but is mentioned here because the mitigation requires bookkeeping changes in the URI Chaser.

URI data structure inefficiency. The Chaser's data structure for storing URIs is a sorted list. Thus, the time complexity for creating a URI list of length n is $O(n^2)$. A malicious RM can waste relying party CPU cycles by simply uploading a large number of certificates with distinct AIA fields. However, this may not have a noticeable effect until the URI list is in the thousands, and even then, other processing may still swamp this. **Mitigation:** If deemed necessary, replace the data structure such that the access/insertion operations which have complexity $O(\log n)$ or better, such as using a red-black tree.

SQL Injection. Since the URIs that arrive in certificates are untrusted data, they may contain special characters that contain SQL commands. **Mitigation:** In the Raytheon BBN implementation, the URI Chaser never uses external data in its queries. Therefore, the URI Chaser component is not vulnerable to SQL injection attacks.

Repository DoS. There can be tens of thousands of relying parties, so it is possible to induce a Smurf-like attack using the relying parties to DoS a repository. A malicious RM wishing to attack targetrepo.com could post a large number of certificates with non-overlapping AIAs

pointing to "`rsync://targetrepo.com/no_such_file.XXXX.cer`" where the X's are replaced with random characters. Even better, some of the URIs could be legitimate published objects. A RP will walk through each of these malicious URIs, establish an rsync connection, and receive an error message saying that the desired path does not exist, but then move on to the next malicious URI to try the same thing. Multiply this by tens of thousands of RPs, and it is reasonable to assume that the bandwidth, the CPU, or the socket descriptors of targetrepo.com would be exhausted. From the perspective of DoS against RPs, this attack wastes RP resources by using the RP to DoS the repository. Perhaps more importantly, this attack causes the target repository to become inaccessible, thereby preventing legitimate certificates from being transferred to relying parties. **Mitigation:** As with Smurf attacks, there is no silver-bullet mitigation against this attack—rather, the relying parties can try to prevent themselves from being *used* to attack the repositories. Since this attack follows the pattern of one repository attacking another repository, it should be feasible to detect the aggressor repositories. For each repository, scan the AIA/SIA/CRLDP fields of all published objects and keep a running total of distinct URIs per domain. If any of these counts becomes abnormally large, especially when accompanied by a large rsync failure rate, the repository should be flagged as suspicious and reported put on a deprecated list pending action by an operator.

Repeated fetching. The current RP software may fetch an object more than once, if that object is published in more than one repository publication point. While this does not affect the AS-IP output of the RPKI database, it does emit a "duplicate object" error message that can distract the operator from true errors. In addition, all URIs are synchronized on each cycle through the RP software. This is unnecessary in some cases. For example, the CRLDP does not need to be synchronized if the current CRL is fresh. Repeated synchronization is not terrible, as rsync will detect that the file has not changed and avoid re-downloading it; however, the unnecessary TCP connection and remote comparison of checksums still consume RP system resources.
Mitigation: A suitable mitigation is still under discussion. In principle, a client-server interaction can ensure that the client has not already seen the hash of a file before downloading that file. However, without modifying rsync internals (undesirable for many reasons), it is difficult to avoid downloading duplicate objects.

Vulnerabilities: Query Client and RTR Server

The Query Client and RTR Server are the two output-generating components of the RP software. The query client reads from the RPKI database and produces RPSL output corresponding to all valid AS-IP associations. The RTR Server periodically creates a snapshot of the RPKI database, and uses the snapshot to answer incremental requests from multiple clients (usually routers) for the latest AS-IP association data.

Minimum Adversary	Vulnerability	Severity	Mitigation Difficulty
Insider RM	Output generation while processing is incomplete	Low	Easy
N/A	Absence of completion indicator	High	Medium
Insider RM	ASN.1-to-C conversion overflows	Low	Easy
Outsider	RTR Server overload	High	Medium

Output generation while processing is incomplete. Consider the following scenario: certificate A, the parent of certificate B, is about to expire. The CA issues replacement certificate C, and the repository atomically replaces A with C. The RP software runs rsync and retrieves the updated file. However, the RP software treats an "update" as a "delete" followed by an "add," so in the time after A is deleted but before C is added, any output will be affected by the fact that B is invalid. This type of anomaly can occur any time output is generated while processing is incomplete. **Mitigation:** Establish SQL table locking to ensure that all semantically atomic actions are in fact atomic and cannot be corrupted by race conditions.

Absence of completion indicator. The RP software does not have a completion indicator. Therefore, it is currently difficult for an operator to gauge system status. **Mitigation:** Create a soft completion indicator, based on the fraction of URIs successfully synchronized, and the status of the DB Updater, DB Garbage Collector, and the URI Chaser. Make this programmatically accessible (e.g., through the database), and create a status interface for the operator.

ASN.1-to-C conversion overflows. RFC 3779 defines an IP address to be an ASN.1 bit string and an AS number to be an ASN.1 integer. Since the ASN.1 types have practically unbounded size ranges, a malicious ROA has the potential of overflowing the C data types. In addition, ASN.1 DER encodings declare data lengths, which in malicious files may not match the actual data length present in the file. **Mitigation:** ASN.1 processing should be made robust to "falsely-advertised" lengths in ASN.1 encodings. Before conversion, the ASN.1 fields should be checked for compliance to the standard ranges of values for IPv4 (32-bit), IPv6 (128-bit), and AS numbers (32-bit).

RTR Server overload. The RTR server handles requests from routers for updates to AS-IP information, using the RPKI/Router protocol [draft-ymbk-rpki-rtr-protocol-05]. It does so by (A) periodically making a snapshot of the RPKI database, and then (B) handling the routers' requests based on the snapshot. By necessity, (A) and (B) are mutually exclusive. Consider the following attack: an outsider compromises a single router that is a client to the RTR server, and then causes it to issue thousands of requests for full copies of the AS-IP information. Since the RTR server is spending all of its CPU resources performing (B), it cannot perform (A), and is thus unable to update its snapshot. This would affect a form of DoS attack on the other routers, since they can no longer obtain up-to-date information. **Mitigation:** While it is difficult to fully defend against this DoS attack, significant improvements can be made. First, the RTR server should give the periodic snapshot of the RPKI database (A) a higher priority than (B), such that the snapshot will be updated periodically regardless of whether router requests are outstanding. Second, the RTR server should rate-limit requests from each authenticated router, so an internal DoS becomes much less likely. Third, the system or network firewall should be configured such that incoming connections to the RTR server are allowed only from authorized routers within the AS served by the RTR server.

Vulnerabilities: DB Updater and DB Garbage Collector

The database updater (rcli) and the database garbage collector perform all of the cryptographic and PKI-related operations. There is significant complexity to these two components, and they

share a significant amount of code. Thus, we begin by describing the algorithms, and then assess vulnerabilities in multiple subsections dedicated to relevant combinations of signed object types.

There are four signed objects of interest: route origination authorizations (ROAs), certificates, certificate revocation lists (CRLs), and manifests. We systematically enumerate vulnerabilities: first those that involve only one type of object, but potentially multiple instances of that object. ROAs and certificates fall in this category. We then enumerate vulnerabilities that involve multiple types of objects. Vulnerabilities related to CRLs and manifests are multi-type, since (1) CRLs are intrinsically tied to certificates, and (2) manifests are intrinsically tied to all three of the other signed object types. We omit the relationship between ROAs and their embedded end-entity (EE) certificates (as well as manifests and their embedded EE certificates), because there are currently no obvious vulnerabilities unique to that simple combination.

Algorithm Description

The BBN RP software views repository changes as a stream of additions and deletions of files, where each file contains a digital object—either a ROA, certificate, CRL, or manifest. The DB Updater is stream-oriented: it processes a single file addition or deletion at a time. In general, if Log Parser reports an added file, the DB Updater verifies the new object to the greatest possible extent, and commits the object along with its validation state (valid or undetermined) to the MySQL database. The DB Updater then recursively propagates state to relevant existing objects: for example, adding a valid parent certificate will propagate validity to any child certificates. If the Log Parser reports a deleted file, the DB Updater does no verification, but deletes the corresponding object in the database, and again propagates state to relevant existing objects.

The following pseudo-code describes the DB Updater's addition operations.

- **Case 1: Add certificate.**
  - Create temporary in-memory certificate object, C_{temp}.
  - Initial object checking. Abort on any failure:
    - Extract X.509 fields from file into C_{temp}: Subject, Issuer, Serial Number, SKI, AKI, SIA, AIA, CLRDP, basicConstraints extension, RFC3779 extensions, notBefore, notAfter.
    - Set FLAG_CA or FLAG_SS (self-signed) in C_{temp}, if applicable.
    - Check X.509 fields in C_{temp} for syntactic compliance with the resource certificate profile [draft-ietf-sidr-res-certs].
  - Relationship checking. Abort if C_{temp} is proved invalid:
    - Define the function fully_valid_parent_cert(X): query the database for the set of certificates P such that $Subject_P = Issuer_X$, $SKI_P = AKI_X$, and $FLAG\_VALID_P = 1$.
    - Search database recursively for fully-validated certificate chain of C_{temp}.

```
Initialize empty stack of certificates, S.
P = fully_valid_parent_cert(C)
While (P is not a trust anchor)
    Push P onto S
    P = fully_valid_parent_cert(P)
    If P is NULL, set FLAG_NOCHAIN in C_temp, defer verification.
T = P
```

Use OpenSSL to verify certificate chain S using trust anchor T, checking signature and RFC3779 extensions.
If valid, set FLAG_VALID in C_{temp}.
Else, C_{temp} has been proved invalid. ABORT.

- Search database for a CRL that revokes C_{temp}.
  Query for valid CRL with Issuer/AKI matching C_{temp}'s Issuer/AKI.
  If CRL is found, search for serial number matching C_{temp}.
  If serial number is listed, C_{temp} has been proved invalid. ABORT.
- Search database for valid manifests containing C_{temp}.
  Query for any valid manifest that lists the path and file for C_{temp}.
  If multiple manifests, use the last one returned by the database.
  If manifest exists, check that file hash for C_{temp} is correct.
  If file hash is correct, set FLAG_ONMAN.
  Otherwise, ABORT.
- Check for duplicate signature for C_{temp}. If found, ABORT.

- Commit C_{temp} to the database, creating database object C_{db}. Note that if FLAG_NOCHAIN was set, then the validity C_{db} is yet undetermined.
- If C_{db} is valid, recursively verify all children in the order: ROAs, CRLs, certificates.
  - If child is ROA, verify and either set FLAG_VALID or delete from database.
  - If child is CRL, verify and either set FLAG_VALID or delete from database. If valid, recursively revoke sibling certificates.
  - If child is certificate, verify and either set FLAG_VALID or delete from database. If valid, recursively verify all children.

- **Case 2: Add CRL.**
  - Create temporary in-memory CRL object, CRL_{temp}.
  - Initial object checking. Abort on any failure:
    - Extract X.509 fields from file into CRL_{temp}: Issuer, AKI, serial number list, CRL number, thisUpdate, nextUpdate.
    - Check X.509 fields for syntactic compliance with the resource CRL profile [draft-ietf-sidr-res-certs].
  - Relationship checking. Abort if CRL_{temp} is proved invalid:
    - Search for fully valid parent certificate, and verify CRL_{temp}:
      P = fully_valid_parent_cert(CRL_{temp})
      If P = NULL, set FLAG_NOCHAIN on CRL_{temp}. Skip verification.
      If P is not NULL, use OpenSSL to verify signature on CRL_{temp}.
      If verification succeeds, set FLAG_VALID.
      Otherwise, CRL_{temp} has been proved invalid. ABORT.
    - Search database for valid manifests containing CRL_{temp}.
      Query for any valid manifest that lists the path and file for CRL_{temp}.
      If multiple manifests, use the last one returned by the database.
      If manifest exists, check that file hash for CRL_{temp} is correct.
      If file hash is correct, set FLAG_ONMAN.
      Otherwise, ABORT.
    - Check for duplicate signature for CRL_{temp}. If found, ABORT.

- Commit CRL_{temp} to the database, creating database object CRL_{db}. Note that if FLAG_NOCHAIN was set, then the validity CRL_{db} is yet undetermined.
- If CRL_{db} is valid, recursively revoke all sibling certificates, and recurse.

- **Case 3: Add ROA.**
  - Create temporary in-memory ROA object, R_{temp}.
  - Initial object checking. Abort on any failure:
    - Extract ROA fields from file to R_{temp}: SKI, IP address information, AS#.
    - Check ROA CMS blob for syntactic compliance with the ROA profile [draft-ietf-sidr-roa-format].
    - Check that the signature on the CMS blob data is consistent with the embedded EE certificate.
  - Extract EE certificate. Abort on any failure:
    - Extract the embedded EE certificate from the ROA, saving it as a file in the same directory as the ROA.
    - Add EE certificate to the database as above, without recursion.
  - Relationship checking.
    - Extract from R_{temp} the (single) AS#, and the digital signature value.
    - Check ROA fields for syntactic compliance with the ROA profile [draft-ietf-sidr-roa-format]. This repeats the above check.
    - Search the database for fully valid parent certificate (the EE certificate), and verify ROA.

      P = fully_valid_parent_cert(R_{temp})
      If P = NULL, set FLAG_NOCHAIN on R_{temp}.
      If P is not NULL, then
        Check RFC3779 extensions against EE's allocation.
        If any prefix or range exceeds EE allocation, ABORT.
        Check signature on ROA contents.
        If signature does not match, ABORT.
        Else signature matches, so set FLAG_VALID.
    - Search database for valid manifests containing R_{temp}.

      Query for any valid manifest that lists the path and file for R_{temp}.
      If multiple manifests, use the last one returned by the database.
      If manifest exists, check that file hash for R_{temp} is correct.
        If file hash is correct, set FLAG_ONMAN.
        Otherwise, ABORT.
    - Check for duplicate signature for R_{temp}. If found, ABORT.
  - Commit R_{temp} to the database, creating database object R_{db}.
  - Cleanup: If ABORT was due to ROA failure, ensure that the EE certificate is removed from the database (algorithm for delete is below).
- **Case 4: Add manifest.**
  - Create temporary in-memory manifest object, M_{temp}.
  - Initial object checking. Abort on any failure:
    - Extract manifest from file to M_{temp}.
    - Check manifest CMS blob for syntactic compliance to manifest definition [draft-ietf-sidr-rpki-manifests].

        - Check that the signature on the CMS blob data is consistent with the embedded EE certificate.
    - Read list of files from M_{temp}, denote as L.
    - Extract EE certificate. Abort on any failure:
        - Extract the embedded EE certificate from the manifest, saving it as a file in the same directory as the manifest.
        - Add EE certificate to the database as above, without recursion.
        - If EE certificate is valid, then set FLAG_VALID on M_{temp}, otherwise set FLAG_NOCHAIN.
    - Commit M_{temp} to the database, creating database object M_{db}.
    - If M_{db} is valid, propagate manifest changes by looping over files and hashes listed by M_{db}:
        - Check actual file hash against manifest hash.
        - If hash matches, set FLAG_ONMAN for the relevant database object.
        - Otherwise, delete the object from the database, and if the file is a certificate, recursively revoke its children.

The following pseudo-code describes the DB Updater's deletion operations.

- **File deletion.**
    - Search the database for the object, X, corresponding to the file/directory.
    - Infer the type based on the object filename extension.
    - If the object is not a certificate, delete it from the database.
    - If the object is a certificate, recursively revoke it and its children as follows:
        - Delete the certificate X from the database, but temporarily cache its SKI and Subject.
        - Search database for all non-self-signed child objects C such that $AKI_C = SKI_X$ and $Issuer_C = Subject_X$. Call this list L.
        - For each child object C in L, count the number of remaining valid parents.
            If the number of valid parents > 0, leave C alone. Remove C from L.
            Else (no valid parent):
                Set C's validity to FLAG_NOCHAIN.
                Search for C's children and append them to L.

The following pseudo-code describes the DB Garbage Collector's operations. Note that the DB Garbage Collector runs periodically, but not concurrently with the DB Updater.

- Check certificate validity intervals.
    - If certificate has a notBefore date in the future, set its FLAG_NOTYET.
    - If certificate is in its validity interval, check FLAG_NOTYET and if set, clear it.
    - If the certificate has a notAfter date in the past, recursively revoke it and children (see above).
- Iterate through all CRLs, recursively revoking any certificates that are listed.
- Check for stale CRLs.
    - If a CRL exists with a nextUpdate in the past, and it has not been superseded by a CRL with a nextUpdate in the future, set all certificates covered by that CRL to FLAG_STALECRL.
- Check for stale manifests.

    - If a manifest exists with a nextUpdate in the past, and it has not been superseded by a manifest with a nextUpdate in the future, set all certificates, CRLs, and ROAs covered by that manifest to FLAG_STALEMAN.
- Check for fresh manifests.
    - For all fresh manifests, i.e., those with a nextUpdate in the future, clear the FLAG_STALEMAN bit (if set) for all certificates, CRLS, and ROAs covered by that manifest.
- Check for fresh CRLs covering certificates in the state FLAG_STALECRL.
    - For any certificates in the state FLAG_STALECRL, if a fresh CRL has arrived with a nextUpdate in the future, then clear the FLAG_STALECRL bit.

Note: As of the time of writing, the BBN RP software handles the most commonly encountered manifest cases, but does not yet perform comprehensive manifest processing.

Route Origination Authorizations (ROAs)

The following vulnerabilities apply identically to all objects, not just ROAs. They are explained here and implied in the other lists.

Minimum Adversary	Vulnerability	Severity	Mitigation Difficulty
Insider RM	Unbounded publication of invalid objects	High	Medium
Insider CA	Unbounded generation of valid objects	High	Hard
Outsider	Known vulnerabilities of supporting libraries	High	Easy
Insider RM	SQL Injection	High	Easy

Unbounded publication of invalid objects. *Note: applies to all objects, not just ROAs.* A malicious repository manager can publish arbitrarily large numbers of signed objects, all of which must be retrieved through rsync. This could cause exhaustion of disk space and CPU of the relying parties, depending on the amount of cryptographic processing needed before objects are deemed invalid. **Mitigation:** As noted in the rsync section, a quota may be placed on the disk space allocated for the local cache corresponding to each repository. Once this quota is exceeded, operator permission is necessary for continued downloading. In addition, the RP software should keep a count of the number of invalid objects per repository. If a large fraction of a repository's published objects are invalid, then the repository should be considered suspect. Downloads from the repository should then be disabled until the operator intervenes.

Unbounded generation of valid objects. *Note: applies to all objects, not just ROAs*. A malicious CA can publish unbounded numbers of fully valid objects with arbitrarily long validity periods. These will exhaust relying party disk space and CPU resources. **Mitigation:** There is no perfect way to prevent this behavior without making assumptions on the number of valid sub-allocations that a CA is permitted to create. For example, in the case of the larger RIRs, thousands of subordinate CAs will be present. One possible approach that will slow the growth process enough to allow continued functionality would be to use a two-tiered limit. The initial synchronization with a repository is bounded by a fixed configured limit. Then subsequent synchronizations can be bounded by a percentage of growth, such as 25% (well above the nominal rate of 5% change per day).

Known vulnerabilities of supporting libraries. *Note: applies to all objects, not just ROAs*. The BBN RP software depends on the following major software packages and libraries: OpenSSL, rsync, MySQL, MySQL/ODBC connector, UnixODBC, and Cryptlib. Any publicly known vulnerability affecting these libraries could allow an outsider or insider threat to compromise the RP software and system. **Mitigation:** Keep each supporting library up-to-date with the latest patches. This requires a small amount of continual software maintenance to adapt to changing library interfaces and functionality.

SQL Injection. *Note: applies to all objects, not just ROAs*. SQL injection can occur when user input is neither strongly typed nor filtered for special characters, and thereby unexpectedly executed. The Raytheon BBN RP software depends on a MySQL database, so it is potentially vulnerable to SQL injection. Most signed objects contain text fields that must be processed and inserted into the database before the object is fully validated. Even after validation, insider attacks are still possible. For example, a certificate's AIA field is an URI that points to a repository file containing the parent certificate. If the URI is maliciously constructed, upon insertion it could corrupt or delete the MySQL database. **Mitigation:** Escape special characters in all external text data destined for the database. There exist standard library routines for scrubbing untrusted inputs to MySQL and other databases. Once Unicode is supported in the Raytheon BBN RP software, it will require additional library processing customized for handling both non-ASCII (e.g. UTF8) byte streams and also ASCII-encoded representations of Unicode.

Certificates

Minimum Adversary	Vulnerability	Severity	Mitigation Difficulty
Insider CA	Certificate validation loops	High	None
Insider CA	Certificate multiple validation path explosion	High	None
Insider RM	Weak duplicate detection allows false collision	High	Easy
Outsider	Unauthenticated network time	High	Easy

Certification validation loops. A loop in a validation chain of certificates could cause a naïve implementation of the RP software to loop infinitely while trying to discover a path to a trust anchor. General X.509 certificates have an extension that could prevent the formation of loops in the validation path: the BasicConstraints Path Length field. However, the Resource Certificate Profile [draft-ietf-sidr-res-certs] prohibits the use of this field. Therefore, the RP path discovery must be robust to certificate loops. **Mitigation:** The Raytheon BBN RP software is already immune to certificate loops because it validates certificates logically from the top-down beginning at the trust anchors. When a new certificate arrives, parent discovery is restricted to certificates that are already validated from a trust anchor. If there is no fully validated parent certificate (whose subject name matches the issuer name of the new certificate, etc.), then the new certificate is simply left in an "unknown" state and awaits further processing.

Certificate multiple validation path explosion. Due to certificate rollover, it is sometimes necessary in practice for a certificate, D, to have two parent certificates: C and C' which are identical except for their validity periods and serial numbers. But if this is possible, then C and C' could share parent certificates B and B', which share parent certificates A and A', etc. A

naïve implementation of RP software might attempt to discover any possible path from certificate D to the trust anchor through a chain of parent certificates, thereby searching an exponentially increasing number of paths: in the example above, there are three levels of two certificates and $2^3 = 8$ potential paths to the trust anchor. **Mitigation:** The BBN RP software is already immune to validation path explosion due to its certificate validation approach. When a new certificate arrives, parent discovery is restricted to certificates that are already validated (to a trust anchor). If there is no validated parent certificate (whose subject name matches the issuer name of the new certificate, etc.), then the new certificate is simply left in an "unknown" state and awaits further processing.

Weak duplicate detection allows false collision. The current unique identifier for signed objects is the object's digital signature value. This is problematic for two reasons. First, the signature value cannot be verified until the parent certificate's public key is retrieved. Second, in the case of RSA signatures (and potentially other algorithms), an adversary who can choose an arbitrary public/private key pair and certificate contents can forge a signature value. This enables the following attack: a malicious RM preempts a valid certificate from another repository by publishing a certificate with the same digital signature. If a relying party is unfortunate enough to download the malicious certificate first, the BBN RP software will reject the second, valid certificate as a duplicate of the first. **Mitigation:** The identification for an object should be made cryptographically strong: either the digital signature *together* with the public key and algorithm identifier, or simply a cryptographic hash of the object. The latter is preferable because it does not require the parent certificate.

Unauthenticated network time. The DB garbage collector handles all time-related state changes. For example, a certificate may expire or enter its validity period, affecting the validity state of all its children, or a CRL or manifest may surpass its "next update" time and become stale, calling into question all sibling certificates. If the network time servers are unauthenticated, then an outsider can impersonate the network time server and advertise a date decades in the future. The RP system clock would be set to the wildly inaccurate date, causing most certificates to expire. **Mitigation:** Configure the NTP client on the RP machine to authenticate its NTP server(s).

Certificate Revocation Lists (CRLs) + associated Certificates

Minimum Adversary	Vulnerability	Severity	Mitigation Difficulty
Insider CA	Large CRL performance penalty	Low	Medium
Insider CA	CRL confinement failure	High	None

Large CRL performance penalty. Under normal RPKI usage, CRLs should not grow very large. However, an insider CA can maliciously publish a CRL that has, for example, 2^{21} serial numbers. Suppose that CA then publishes 2^{12} certificates that are siblings of this malicious CRL. The current software will attempt to compare each new certificate with each entry on the CRL, resulting in 2^{33} comparisons, a significant performance hit. **Mitigation**: Fortunately, RFC 5280 states: "A complete CRL lists all unexpired certificates, within its scope, that have been revoked for one of the revocation reasons covered by the CRL scope." Under normal RPKI usage, a standards-compliant CRL will not legitimately list expired certificates, nor grow to be orders of

magnitude larger than the number of sibling certificates. Therefore, this vulnerability can be mitigated in the following manner. Before a certificate/CRL search is done, first compare the number of sibling certificates to the length of the CRL—an operation that can be done efficiently as a database query followed by a numerical comparison. Except during initial synchronization, if the CRL length is significantly larger than the number of certificates, the CRL and the responsible CA should be marked as suspicious and any related objects put on hold for manual approval.

CRL confinement failure. A CRL should not be allowed to revoke any certificates other than direct sibling certificates issued by its parent CA. The RP software implementation should enforce this, and check *both* the Issuer Name and the AKI of the CRL, so that a CRL cannot be signed by one issuer (AKI), yet claim a second issuer (Issuer Name), and therefore revoke certificates signed by the second issuer. **Mitigation**: None needed. The Raytheon BBN RP software safely checks both fields. While this vulnerability is important to check, the fact that CRLs have been well-established for some time means that most implementations are likely to perform CRL processing correctly.

Manifests + associated Certificates, CRLs, and ROAs

Minimum Adversary	Vulnerability	Severity	Mitigation Difficulty
Insider CA	Large manifest performance penalty	Low	Easy
Insider CA	Manifest confinement failure	High	Easy
Insider CA	Manifest conflicts	High	Easy
Insider RM	Incomplete manifest corner-case implementation	High	Medium/Hard

Large manifest performance penalty. Like the large CRL performance penalty (see above), there exists a large manifest performance penalty. A malicious CA may publish an abnormally long manifest, causing the RP software waste system resources on extraneous checking. **Mitigation:** Large manifests are easier to detect than large CRLs, because manifests must contain a list of items with a one-to-one correspondence with the actual signed objects that have been downloaded from a repository directory. Therefore, the RP can place much more stringent requirements on a manifest's size, and check that the number of files in the directory is not dwarfed by the length of the manifest.

Manifest confinement failure. A manifest is signed by a CA, yet governs the contents of a repository directory. CAs and RMs can be independent entities, and repository layouts are not required to conform to the certificate issuance hierarchy. Because of this, a manifest is considered authoritative only for the *subset* of files in the directory that were issued by its parent CA. On the other hand a malicious CA could attempt to *claim* files issued by other CAs. **Mitigation:** Disallow manifests from claiming files in directories other than the current one. The case of conflicting manifests in the same directory is handled below.

Manifest conflicts. Due to key and algorithm rollover scenarios, it is necessary for the RP software to allow multiple manifests in the same directory, potentially signed by different CA instances (old and new, relative to a CA rekey). Unfortunately, this opens the potential for a malicious CA to create a conflicting manifest in the same directory that lists a file already on the

legitimate manifest. **Mitigation**: A manifest is validated using an EE certificate issued under a CA, and that same CA is the parent of the certificates in the manifest. Therefore, if two CAs issue a manifest claiming a certificate foo.cer, only the CA that issued foo.cer is authorized to place it on the manifest. Assuming foo.cer has been validated up to at least its direct parent CA, the legitimate manifest can be determined and the other manifest should be marked as suspicious and reported to the operator.

Incomplete manifest corner-case implementation. There are currently many manifest states that are not handled by the RPKI software, partly due to lack of standard agreement about the correct semantics. Mitigation: Implement reasonable manifest semantics, adapting the RP software as the IETF working group converges on a standard for manifest behavior.

Vulnerabilities: Server Configuration

Minimum Adversary	Vulnerability	Severity	Mitigation Difficulty
Outsider	MySQL server insecurity	High	Easy
Outsider	Firewall insecurity	High	Easy
Outsider	Unauthenticated network time	High	Easy

MySQL server insecurity. There are standard vulnerabilities that come with many default installations of MySQL, such as password-less root logins, remote access, etc. **Mitigation:** Use the standard methods to secure the MySQL installation, and disable any accesses which are unnecessary. A bonus would be to create limited MySQL accounts for components that do not require write access to the main database tables, such as the URI Chaser, the Query Client, and the RTR Server.

Firewall insecurity. Early in the design of the Raytheon BBN RP software, it was conceived that the Local Repository and rsync processes may occur on a machine separate from the MySQL database and its clients. Therefore, the Log Parser and the DB Updater communicate on an unauthenticated TCP socket which could be hijacked. **Mitigation:** Ensure that the components are in a secure enclave: that the system firewall prevents outside access to the ports used by the RP software.

Unauthenticated network time. As mentioned before, an outsider who can modify NTP traffic can cause RP system time changes that change the validity of potentially all signed objects. **Mitigation:** Choose a set of authenticated NTP servers and enable authentication checking in the RP machine's NTP client.

Conclusion

A properly functioning RPKI requires wide deployment of secure relying party software that can deliver valid, timely route origination information to routers, even in the presence of motivated and capable adversaries. The Raytheon BBN RP software provides a robust and efficient architecture for retrieving and verifying the information published in the RPKI repository system. While the current Raytheon BBN implementation has a number of denial-of-service vulnerabilities, most of the vulnerabilities have standard mitigations involving proper input sanitization and enforcing the principle of least privilege in different contexts. In more difficult

cases where DoS attacks cannot be perfectly prevented, the mitigations involve the detection of malicious activity, followed by an automated "do no harm" default mitigation, followed by alerting of an operator. In a few cases involving repository authentication and the manifest/repository relationship, this DoS assessment has revealed specification-level security issues that need to be discussed by the IETF Working Group.

References

Austein R, Huston G, Kent S, Lepinski M. Manifests for the Resource Public Key Infrastructure. *Internet-Draft*. Available at: http://tools.ietf.org/html/draft-ietf-sidr-rpki-manifests-07.

Bush R, Austein R. The RPKI/Router Protocol. *Internet-Draft*. Available at: http://tools.ietf.org/html/draft-ymbk-rpki-rtr-protocol-05.

Cooper D, Santesson S, Farrell S, et al. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. *RFC 5280*. Available at: http://tools.ietf.org/html/rfc5280.

Housley R. Cryptographic Message Syntax (CMS). *RFC 5652*. Available at: http://tools.ietf.org/html/rfc5652.

Huston G, Loomans R, Michaelson G. A Profile for Resource Certificate Repository Structure. *Internet-Draft*. Available at: http://tools.ietf.org/html/draft-ietf-sidr-repos-struct-04.

Huston G, Michaelson G, Loomans R. A Profile for X.509 PKIX Resource Certificates. *Internet-Draft*. Available at: http://tools.ietf.org/html/draft-ietf-sidr-res-certs-18.

Kent S. How Many Certification Authorities are Enough? In: *Proceedings of MILCOM 97*. IEEE Press; 1997:61-68.

Lepinski M, Kent S. An Infrastructure to Support Secure Internet Routing. *Internet-Draft*. Available at: http://tools.ietf.org/html/draft-ietf-sidr-arch-09.

Lepinski M, Kent S, Kong D. A Profile for Route Origin Authorizations (ROAs). *Internet-Draft*. Available at: http://tools.ietf.org/html/draft-ietf-sidr-roa-format-06.

Lynn C, Kent S, Seo K. X.509 Extensions for IP Addresses and AS Identifiers. *RFC 3779*. Available at: http://tools.ietf.org/html/rfc3779.

Michaelson G, Kent S, Huston G. A Profile for Trust Anchor Material for the Resource Certificate PKI. *Internet-Draft*. Available at: http://tools.ietf.org/html/draft-ietf-sidr-ta-04.

Montana D, Reynolds M. *Validation Algorithms for a Secure Internet Routing PKI*.; 2006. Available at: http://vishnu.bbn.com/papers/europki.pdf .

Reynolds M, Kent S. Local Trust Anchor Management for the Resource Public Key Infrastructure. *Internet-Draft*. Available at: http://tools.ietf.org/html/draft-reynolds-rpki-ltamgmt-00.

Reynolds M. Final Report: PKI for Internet Address Space, Contract FA8750-07-C-0006. 2007.