Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 30 November 2026

B. Mottershead
29 May 2026

Resolvable Universally Unique Identifiers (RUUID)
draft-motters-ruuid-00

Abstract

This document defines Resolvable Universally Unique Identifiers (RUUIDs): a UUID format encoding a 64-bit IPv6 network prefix, a 48-bit identifier, and a 10-bit type. A resolver uses reverse DNS on the network prefix to discover the associated domain and, via that domain's "UUID document", constructs a URI of the referent. Defaults for both the document location and the referent URI template mean that when URLs follow the canonical convention, nothing beyond a reverse-DNS PTR record needs to be published; resolution degrades gracefully when configuration is missing.

The 64-bit network field carries an IPv6 /64 prefix directly. An IPv4 /32 is encoded as the corresponding 6to4 prefix (RFC 3056). RUUIDs are 128 bits in the standard UUID textual form. Pending a dedicated UUID version, they use the RFC 9562 experimental version 8 with variant 10, so existing UUID parsers recognise them as well-formed UUIDs.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 30 November 2026.

Table of Contents

1. Introduction

UUIDs as defined in [RFC9562] are opaque: a UUID names a thing but says nothing about where to learn more about it. This document defines a _Resolvable UUID_ (RUUID) that binds the UUID at generation time to an IP network prefix. Reverse DNS resolves that prefix to a domain, and a "UUID document" resolves the domain and the fields of the UUID to a URI of its referent, allowing the UUID to be dereferenced.

The bytes of an RUUID fully specify the algorithm a resolver MUST follow; there is no fallback or probing. Resolution uses only DNS, which may include DNS-over-HTTPS, with no new infrastructure required.

An RUUID parses as a normal UUID with a well-defined version and the RFC 9562 variant; code that does not understand RUUIDs SHOULD treat them as opaque.

1.1. Relation to other identifier systems

RUUID builds on preexisting UUID versions and variants, and relates to several other identifier systems. The relevant prior art includes:

* Preexisting UUID versions and variants ([RFC9562]), which provide coordination-free generation and enduring universal uniqueness, but whose identifiers are opaque: a UUID names a referent without indicating where to find it. An RUUID is itself a UUID (currently version 8 with variant 10) that adds resolvability to these properties, through a public and reproducible resolution method.

* URNs ([RFC8141]) are URIs urn:NID:NSS, requiring an IANA-registered Namespace Identifier per namespace; resolution is not standardized at the URN layer and varies per namespace. [RFC4122] defined a URN namespace, urn:uuid:, for UUIDs.

* W3C Decentralized Identifiers (DIDs, [W3C-DID]) are URIs did:METHOD:ID with per-method registration and per-method resolution rules; an RUUID can be expressed as a DID via the did:uuid method, which is specified separately.

* Numerous numbering schemes, such as IEEE Extended Unique Identifiers (EUI), Universal Product Codes (UPC), International Article Numbers (EAN), International Standard Book Numbers (ISBNs), Vehicle Identification Numbers (VINs), Object Identifiers (OIDs), Archival Resource Keys (ARKs), Handle-based identifiers ([RFC3650]), and Digital Object Identifiers (DOIs) depend on registration with an assignment authority. Many of these are resolvable but depend on dedicated resolution infrastructure.

RUUIDs differ from the prior art in the following ways.

* Unlike many of the existing numbering schemes, assignment of UUIDs in general, and RUUIDs in particular, does not depend on a central registration authority. By following the specification, anybody may generate a UUID and assign it to a referent with high confidence of enduring universal uniqueness, without dependence on authority or coordination with other parties generating UUIDs. Accordingly, UUIDs from independent sources may be pooled in databases and datastreams, without coordination.

* They are 128-bit UUIDs in the conventional textual form (Section 4 of [RFC9562]), not URIs. Code that does not know this specification can still parse, store, and compare them as UUIDs.

* Resolution of RUUIDs runs over the existing RIR / LIR / ISP allocation chain and the DNS reverse-zone delegation chain, with no dedicated registry or central resolver. The accountability gradient is the one that already governs IP address space.

1.2. Applicability

As an _identifier_, an RUUID unambiguously names its referent indefinitely, like any UUID. As a _resolvable_ identifier, it can be walked through DNS to the referent’s URI which can de-referenced; this role is bounded by continued control of the network prefix and its reverse-DNS delegation (see Section 9.2).

Workflows requiring coordination-free generation of structured (sortable, time-ordered) identifiers SHOULD prefer UUID version 7 ([RFC9562]); see Section 7.4.

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are used in this document:

RUUID A 128-bit identifier conforming to this specification.

Network prefix The 64-bit field of an RUUID that carries an IPv6 /64 network prefix. An IPv4 /32 is encoded as the corresponding 6to4 prefix ([RFC3056]); see Section 3.2.

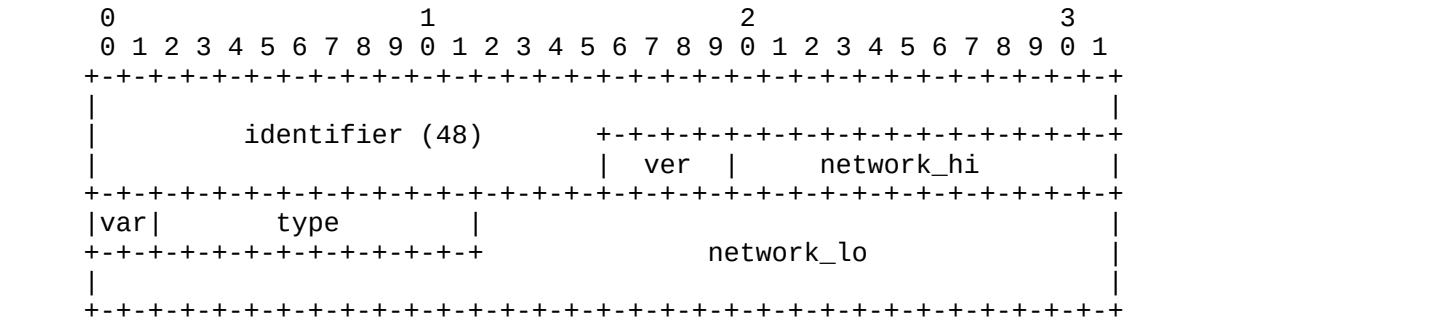
Identifier The 48-bit field of the RUUID that names the referent within the network prefix. See Section 3.

UUID document A JSON document, identified by a URI published in DNS at _uuid.<domain>, that describes how RUUIDs under a network prefix are generated, and optionally publishes cryptographic material for verifying them. The format is defined in Section 6.

3. RUUID Format

3.1. Bit layout

Treating the RUUID as a 128-bit big-endian unsigned integer with bit 127 being the most significant bit:



field	width	meaning
identifier	48	identifier of the referent within (network, type)
version	4	UUID version, set to 8
network_hi	12	high 12 bits of the 64-bit network prefix
variant	2	UUID variant, set to binary 10
type	10	identifier of a type entry in the UUID document
network_lo	52	low 52 bits of the network prefix

Table 1

3.2. The network prefix

The 64-bit _network prefix_ is the concatenation of network_hi and network_lo, treated as a single 64-bit big-endian unsigned integer. It is an IPv6 /64 network prefix. An IPv4 /32 is encoded as the corresponding IPv6 6to4 prefix ([RFC3056]), zero-padded on the right; so a single 64-bit field carries either address family; the encoding distinguishes them.

Note that while [RFC3056] [RFC7526] deprecated the companion anycast-relay prefix, the 6to4 IPv6 prefix is still allocated, and no native IPv6 addresses are assigned from it. Each /64 within 2002::/16 corresponds to a specific IPv4 /32 under the 6to4 derivation, and only that IPv4 owner controls the corresponding in-addr.arpa zone. A network field beginning with 0x2002 MUST be interpreted as IPv4-encoded; an RUUID with a network field in 2002::/16 MUST encode an IPv4 /32 whose in-addr.arpa reverse zone is controlled by the RUUID’s resolver.

The network prefix is an administrative identifier, not the IP address of any specific service; any number of servers MAY share a single network prefix (see Section 7.1).

3.3. Version and variant

The version field MUST be 8 (RFC 9562 "experimental"); the UUID variant field MUST be binary 10.

3.4. Type

The 10-bit type field identifies a type entry in the UUID document (Section 6). The 1024 type values each describe one kind of referent and can carry their own referent-URI template. Type entries appear in the UUID document as CID service entries with id = #<type> (see Section 6.2). Type 0 is reserved for "unspecified".

4. Resolution

4.1. Overview

RUUID resolution is two-phase. *Phase 1* resolves the network field to a UUID document (Section 6). *Phase 2* applies the document's per-type service template to the 48-bit identifier to produce the referent URI. The spec's normative output is the referent URI.

A resolver executes:

A. *Phase 1*

1. Determine address family. If the top 16 bits of the network field equal 0x2002, the next 32 bits are the IPv4 /32 (reverse zone in-addr.arpa); otherwise the full 64 bits are an IPv6 /64 network prefix (reverse zone ip6.arpa).

2. Consult a _registry endpoint_ (Section 4.2) for the domain and UUID-document URI associated with the network prefix. This is the only step that MUST succeed; later steps have documented defaults.

3. Fetch the UUID document (Section 6). On fetch failure or non-JSON body, proceed with no document.

B. *Phase 2*

1. Construct the referent URI: pick the service entry from the UUID Document which corresponds to the RUUID's type (or fall back per Section 4.5) and apply placeholder substitution.

Per-identifier metadata lives in the UUID document, not the registry, so registry state per domain is constant regardless of how many RUUIDs are generated under that domain.

4.2. Registry endpoints

A registry endpoint returns the domain and UUID-document URI for a given network prefix, reached over DNS. The resolver does a PTR query against the reverse-DNS name (Section 4.3) for the domain, then a URI query at _uuid.<domain> (falling back to TXT; Section 4.4) for the document URI. The system DNS resolver is the default; any DNS-protocol-speaking service with these semantics is conformant. The DNS resolver may be either traditional DNS over UDP/TCP port 53 or a DNS-over-HTTPS (DoH) service ([RFC8484]).

Every public IP prefix has a reverse-DNS chain, so resolution needs no dedicated registry or new infrastructure.

4.2.1. Endpoint configuration

A registry endpoint is configured by URL; the scheme determines how it is reached. A DNS-protocol endpoint uses dns://<host>[:<port>] ([RFC4501]; port defaults to 53). A DoH-reachable endpoint uses an https:// URL ([RFC8484]), carrying wire-format DNS messages (application/dns-message).

4.3. Reverse-DNS name construction

For IPv4-encoded RUUIDs (top 16 bits = 0x2002), recover the IPv4 /32 as (network >> 16) & 0xFFFFFFFF and construct the reverse-DNS name per Section 3.5 of [RFC1035]: the four address bytes in reverse order as decimal labels, followed by in-addr.arpa.

For IPv6-encoded RUUIDs, the reverse-DNS name is the standard ip6.arpa name (Section 2.5 of [RFC1035]) truncated to the first 16 nibbles (the /64 boundary).

4.4. UUID document resolution

At _uuid.<domain> the resolver issues a URI query first ([RFC7553]), then a TXT query (Section 5) if no usable URI record is returned. With multiple URI records, pick the lowest priority, breaking ties by highest weight. TXT records without the RUUID prefix MUST be ignored.

Type-specific queries are used rather than ANY because [RFC8482] permits recursive resolvers to return minimal answers to ANY meta-queries, silently losing published records.

4.4.1. Default document URI

If no usable record is present at _uuid.<domain>, the UUID-document URI defaults to https://<domain>/.well-known/uuid-document.json (well-known suffix per [RFC8615], registered in Section 8). On a fetch failure of the default URI, the resolver applies the default referent template (Section 4.5.2). A UUID document hosted elsewhere is located by publishing a URI or TXT record at _uuid.<domain>.

4.5. Referent URI construction

The referent URI is constructed by selecting a template and applying placeholder substitution.

4.5.1. Template selection

The resolver picks a template by checking, in order:

1. A service entry with id = #<type> and a string serviceEndpoint (the type-specific template).

2. A service entry with id = #0 and a string serviceEndpoint (the domain-wide default).

3. The spec-wide default template defined below.

4.5.2. Default template

When no document template applies, the default is:

https://<domain>/<type>/<identifier>

The default applies when the service array is absent, has neither a #<type> nor #0 entry, or has only entries without serviceEndpoint; it also applies when the document cannot be fetched at all (so reverse-DNS success alone always yields a resolvable URL). When URLs already follow this convention, no UUID document need be published.

4.5.3. Placeholder substitution

The resolver replaces every occurrence of the following substrings:

placeholder	substituted with
<identifier>	when <day> is absent from the template, the full 48-bit identifier as 12 lowercase hex digits, zero-padded; when <day> is present, the low 28 bits (the sequence of Section 7.3) as 7 lowercase hex digits, zero-padded
<day>	the high 20 bits of the identifier (the day_count of Section 7.3), rendered as the corresponding calendar date YYYY-MM-DD (2025-01-01 + day_count days, UTC)
<type>	the RUUID’s type as a decimal integer, no padding
<network>	the network field as 16 lowercase hex digits, zero-padded
<uuid>	the full RUUID in canonical 8-4-4-4-12 lowercase form
<domain>	the domain returned by the PTR lookup

Table 2

Substituted values are not URI-encoded; all are URL-safe by construction.

A template containing <day> is an assertion by the document publisher that identifiers under this entry are constructed per Section 7.3. The resolver does not check this and performs the substitution as defined regardless of the underlying bits; a <day> value implausible for Section 7.3 (e.g., a date far in the future) is the publisher’s misconfiguration, not the resolver’s concern.

The result MUST be a valid URI reference per Section 4 of [RFC3986]. A relative reference is resolved against the UUID document’s fetch URI per Section 5 of [RFC3986]. UUID documents SHOULD use *absolute-path references* (a path beginning with /, no scheme, no authority) so the document can be moved between hosts without editing service entries: /-rooted references transplant the document’s scheme and authority unchanged.

5. TXT Record Format

When the _uuid.<domain> query returns no URI records but does return TXT, the resolver picks a TXT record whose concatenated strings (Section 3.3.14 of [RFC1035]) match:

v=ruuid1 <URI>

<URI> MUST be a valid URI per Section 3 of [RFC3986].

6. UUID Document

The UUID document is the JSON document fetched at step 3 of Section 4. It is a W3C Controlled Identifiers (CID) document ([W3C-CID]) whose service array carries per-type referent-URI templates. Resolvers MUST ignore fields they do not recognise, and SHOULD treat the document as authoritative only for the domain whose DNS pointed them at it.

6.1. Document fetch

The UUID document is identified by the URI established at step 3 of Section 4: either an explicitly-published URI from _uuid.<domain> or the default (Section 4.4.1). The URI MAY use any scheme the resolver can dereference to a fetchable JSON document: https: (the expected baseline), http:, data: (inline JSON), file:, or did: (where the DID resolves to a document that _is_ the UUID document; did:web:, did:plc:, and did:uuid: are all usable). UUID documents SHOULD use https: for TLS transport integrity.

The document MUST be UTF-8 JSON ([RFC8259]) with an object at top level.

6.2. Document structure and evolution

A conforming UUID document is a JSON object structured as a W3C Controlled Identifiers (CID) document [W3C-CID]. It is the CID document for the network prefix’s controller: one document, keyed by the network prefix, covers every full RUUID under that network prefix.

The UUID document’s id is the URI used to dereference it (CID canonical-URL): the URI published at _uuid.<domain>, or the default-document URI (Section 4.4.1) when no record is published. The URI MAY use any URL-Standard scheme.

The fields used by the resolution pipeline are:

field	type	required by this spec	meaning
@context	string or array	optional (recommended)	JSON-LD context. SHOULD include "https://www.w3.org/ns/cid/v1".
id	string	optional (recommended)	The URI at which the document was dereferenced.
controller	string or array	optional	URI(s) of the entity authorised to update this document. When omitted, CID treats it as equal to id.
alsoKnownAs	array of strings	optional	Additional URIs identifying the same subject.
service	array of objects	required for type entries	Per-type referent templates; see Section 6.4.

Table 3

No top-level field is REQUIRED: the empty object {} is processable (it just makes no RUUIDs resolvable, other than those resolvable using the default template). The schema evolves by addition; resolvers MUST ignore unknown fields. A resolver that needs a field which is absent or unparseable MUST fail that operation rather than substitute defaults.

6.3. Verification fields

A UUID document MAY include prefixes (array of CIDR strings) and domains (array of domain strings) covered by the document. Consumers SHOULD verify the RUUID’s network prefix against prefixes and the resolved domain against domains when present, treating a mismatch as misconfiguration (log; MAY refuse the referent URI). These fields are advisory, not security-critical: an adversary who controls the DNS path or the document can set them to any value. Their purpose is detecting honest mistakes (staging docs at prod URLs, shared infrastructure pointing at the wrong tenant, a prefix move without re-hosting the document).

6.4. Service entries (per-type referent templates)

The document's service array carries one entry per RUUID type covered by the document. Entries follow the CID service-entry schema with one convention: the entry's id is the fragment URI #<type> (the decimal type value, "0" through "1023"), and serviceEndpoint is the referent-URI template the resolver substitutes (Section 4.5).

```
{
  "@context": "https://www.w3.org/ns/cid/v1",
  "id": "https://example.com/.well-known/uuid-document.json",
  "alsoKnownAs": ["did:uuid:00000000-0000-8200-8002-c000022a0000"],
  "service": [
    {
      "id": "#1",
      "type": "User",
      "serviceEndpoint": "https://example.com/u/<identifier>"
    },
    {
      "id": "#42",
      "type": "CowTag",
      "serviceEndpoint": "https://example.com/t/<identifier>"
    }
  ]
}
```

The array is sparse; missing entries fall back to the default template (Section 4.5.2). Different types MAY use entirely different URI structures.

6.4.1. Type entry fields

field	meaning
id	the fragment URI #<type> (e.g. #1, #42); the type value is the numeric string after #
type	arbitrary string or set of strings.
serviceEndpoint	URI template per Section 4.5; the default template applies if absent

Table 4

The id field is OPTIONAL in the Controlled Identifier specification, and may be any valid URI that is unique within the list of services. However, a RUUID resolver SHOULD ignore service entries with no id, and service entries whose id does not have a numeric fragment between #0 and #1023.

The Controlled Identifier specification requires the type field, but the RUUID resolver makes no use of it. A resolver MUST NOT treat a missing type field as an error, even though a service entry lacking type does not conform to the Controlled Identifier specification. If present, a resolver MAY return the type value as additional information.

The serviceEndpoint field is REQUIRED in the Controlled Identifier specification. When the RUUID resolver selects a service entry because its id fragment matches the RUUID's type value, the serviceEndpoint is the template used to construct the URI of the referent.

7. Generation Considerations

7.1. Network prefix semantics

The network field is an IPv6 /64 network prefix, or an IPv4 /32 6to4-encoded into the same field. It is not the IP address of any specific service. The resolution algorithm requires only that the prefix's reverse-DNS zone resolve PTR queries to the domain that publishes the UUID document; the operator of that DNS zone need not coincide with the entity that generated the RUUID nor with the operator of the service providing the referent. Any number of servers, on unrelated addresses, MAY share a single network prefix.

7.2. Uniqueness

Identifiers MUST be unique within (network, type). Global uniqueness of RUUIDs follows from this property and the global uniqueness of IP allocations, provided the generating entity has control of the network prefix at generation-time and maintains that indefinitely. Once an identifier is assigned, the binding MAY be retired but the identifier value MUST NOT be reused for a different referent.

7.3. Identifier construction

For an RUUID used only within the generator's local scope, the 48-bit identifier MAY be any value satisfying Section 7.2.

For an RUUID used outside the generator's local scope, where persistent universal uniqueness is required without ongoing tenancy by the generator of the network prefix, and without ongoing coordination among independent generators, the identifier SHOULD be constructed as follows:

identifier = (day_count << 28) | sequence

* day_count is a 20-bit unsigned count of days since 2025-01-01 00:00:00 UTC. The value MUST be a day on which the generator held reverse-DNS authority over the network prefix, and MUST NOT be a future day. Any day satisfying these two conditions is valid; day_count need not be the day on which the RUUID is generated, and an implementation MAY reuse the same day_count for many RUUIDs (e.g., the earliest day of its tenure of the network id, advancing only when the 2^28 sequence space is exhausted). The count wraps after 2^20 days (approximately year 4896).

* sequence is a 28-bit value unique within (network, type, day_count).

Successive controllers of a network prefix hold disjoint sets of tenure days, so the two MUST conditions above guarantee that their day_count values do not overlap, and identifiers from different controllers cannot collide even without coordination between them.

This yields 2^28 = 268,435,456 RUUIDs per (network, type, day) and, across the 1024 type values, approximately 275 billion RUUIDs per network prefix per day of tenancy.

The sequence uniqueness requirement is scoped to a single day within a single (network, type); coordination across days, across types, and across network prefixes is not required. Implementations MAY satisfy it with a counter, a time-of-day component with within-period randomness, deliberate partitioning across cooperating generators, or any other scheme. Random selection from the 28-bit space achieves a 2^-14 collision probability at approximately 16,000 RUUIDs per (network, type, day) and degrades sharply at higher volumes; coordination between multiple generator "workers" of some form is required to approach the per-day maximum.

Resolvers MUST NOT interpret this structure. RUUIDs constructed this way are byte-indistinguishable from RUUIDs with opaque identifiers; the convention is a generator-side discipline that helps cooperating generators avoid collisions, not a resolution input.

7.4. Collisions

The 48-bit identifier yields a ~2^24 birthday bound for random identifiers within a single (network, type) tuple, which provide uncoordinated collision avoidance via randomness. The "uniqueness durability" of RUUIDs may be less in some cases than with other UUID versions, such as those based on hashes, timestamps, or pseudo-random number generators. For example, RUUIDs generated after control of network id has passed to a different entity may be more prone to collision than other UUID versions, if they are pooled in the same databases or datastreams with RUUIDs from a previous controller of the network id.

8. IANA Considerations

This document requests:

1. Assignment of a dedicated UUID version to this specification, provisionally requested as version 9 (or, if 9 is unavailable, the next unassigned UUID version). See [RFC9562]. Pending such allocation, implementations use the RFC 9562 experimental version 8.

2. Establishment of an "RUUID TXT Prefix Registry" administered under IETF Review [RFC8126], with the initial entry v=ruuid1 referencing this document.

3. Registration of uuid-document.json in the "Well-Known URIs" registry per [RFC8615], referencing this document. The well-known URI provides the default UUID-document location defined in Section 4.4.1.

9. Security Considerations

9.1. Registry endpoint trust

RUUID resolution inherits its registry endpoint’s trust substrate: a DNS-protocol endpoint inherits the DNS hierarchy (DNSSEC, where deployed, authenticates the _current_ operator rather than continuity across transfers; see Section 9.2). A DoH-reachable endpoint additionally relies on the TLS connection to the configured URL ([RFC9525]). Strengthening trust beyond that substrate (signed records, externally rooted continuity evidence) is out of scope.

9.2. Prefix transfer and long-term durability

An RUUID's identifier role is unconditional, but its resolution role depends on continued control of the network prefix and its reverse-DNS delegation. IP prefixes can be transferred between operators; DNS provides no notion of _continuity of identity_ across a transfer (a new operator configuring reverse DNS competently is indistinguishable from the original, even with DNSSEC).

Three observable states at Phase 1:

Still valid. PTR returns the intended domain; the binding is intact.

Detectably rotted. The prefix is no longer maintained for RUUID purposes (NXDOMAIN, or a PTR to a domain serving no UUID document). The resolver gets a clear failure signal, better than plain UUIDs (no resolution at all) or a lapsed HTTPS URL (indistinguishable 404).

Commandeered. The prefix has been reassigned and the new operator serves a PTR of their choosing. The resolver sees an authentic (DNSSEC-validatable) PTR leading to an unintended domain. This is the genuine new failure mode RUUID introduces; mitigating it generally is an open problem for the IETF and address-registry community.

Network prefixes used in RUUIDs SHOULD be ones expected to be held for the identifiers' operational lifetime; native IPv6 from a durably held PA/PI allocation is the strongest choice. The 6to4-encoded IPv4 form inherits IPv4-transfer risk. Consumers SHOULD record the PTR target at first resolution and compare on later resolutions; this is trust-on-first-use, with the usual caveat that it offers nothing to a consumer whose first encounter is after the change.

Phase 2 is plain URI resolution. This spec places no constraint on the URI scheme; RUUID inherits whatever durability properties the scheme provides. The common case is HTTPS, with the same partial trust model that governs every HTTPS link on the web.

Even when resolution rots or is commandeered, the RUUID remains a valid identifier for its referent in the consumer's own records.

9.3. Privacy

An RUUID discloses a network prefix associated with its referent. This is generally similar in sensitivity to the domain itself, but in some configurations may be more revealing (e.g., a customer prefix inside a hosting provider). Consider this when choosing the network prefix for an RUUID.

When the recommended construction (Section 7.3) is used, the RUUID's day_count discloses a day on which the generator held the network prefix, information of similar character to public WHOIS and reverse-DNS delegation records. To suppress even this signal, use an opaque 48-bit identifier instead, subject to Section 7.4.

9.4. Identifier collision

RUUIDs do not provide cryptographic uniqueness guarantees beyond what the chosen identifier scheme provides. RUUIDs generated with small counter-style identifiers are vulnerable to identifier guessing; when this is a concern, use random identifiers filling the available width.

9.5. DoS surface

Adversarial RUUIDs could direct registry queries at arbitrary network prefixes and HTTPS fetches at arbitrary URLs. Resolvers SHOULD apply standard DNS rate-limiting and HTTP client limits, and SHOULD cache UUID documents per domain so mass ingestion from one domain does not multiply fetches.

10. References

10.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

[RFC7553] Faltstrom, P. and O. Kolkman, "The Uniform Resource Identifier (URI) DNS Resource Record", RFC 7553, DOI 10.17487/RFC7553, June 2015, <https://www.rfc-editor.org/info/rfc7553>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

[RFC9525] Saint-Andre, P. and R. Salz, "Service Identity in TLS", RFC 9525, DOI 10.17487/RFC9525, November 2023, <https://www.rfc-editor.org/info/rfc9525>.

[W3C-CID] "Controlled Identifiers v1.0", W3C Working Draft, n.d., <https://www.w3.org/TR/cid-1.0/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

10.2. Informative References

[RFC3056] Carpenter, B. and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", RFC 3056, DOI 10.17487/RFC3056, February 2001, <https://www.rfc-editor.org/info/rfc3056>.

[RFC7526] Troan, O. and B. Carpenter, Ed., "Deprecating the Anycast Prefix for 6to4 Relay Routers", BCP 196, RFC 7526, DOI 10.17487/RFC7526, May 2015, <https://www.rfc-editor.org/info/rfc7526>.

[RFC4501] Josefsson, S., "Domain Name System Uniform Resource Identifiers", RFC 4501, DOI 10.17487/RFC4501, May 2006, <https://www.rfc-editor.org/info/rfc4501>.

[RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <https://www.rfc-editor.org/info/rfc8484>.

[RFC8482] Abley, J., Gudmundsson, O., Majkowski, M., and E. Hunt, "Providing Minimal-Sized Responses to DNS Queries That Have QTYPE=ANY", RFC 8482, DOI 10.17487/RFC8482, January 2019, <https://www.rfc-editor.org/info/rfc8482>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/info/rfc8141>.

[RFC3650] Sun, S., Lannom, L., and B. Boesch, "Handle System Overview", RFC 3650, DOI 10.17487/RFC3650, November 2003, <https://www.rfc-editor.org/info/rfc3650>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[W3C-DID] "Decentralized Identifiers (DIDs) v1.0 -- Core architecture, data model, and representations", W3C Recommendation REC-did-core-20220719, 2022, <https://www.w3.org/TR/did-core/>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

Author’s Address

Brian Mottershead
Email: bmotters@gmail.com