Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 25, 2013

D. Thaler, Ed.
Microsoft

August 24, 2012

Issues in Identifier Comparison for Security Purposes
draft-iab-identifier-comparison-04.txt

Abstract

Identifiers such as hostnames, URIs, and email addresses are often used in security contexts to identify security principals and resources. In such contexts, an identifier supplied via some protocol is often compared against some policy to make security decisions such as whether the principal may access the resource, what level of authentication or encryption is required, etc. If the parties involved in a security decision use different algorithms to compare identifiers, then failure scenarios ranging from denial of service to elevation of privilege can result.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

In computing and the Internet, various types of "identifiers" are used to identify humans, devices, content, etc. Before discussing security issues, we first give some background on some typical processes involving identifiers.

As depicted in Figure 1, there are multiple processes relevant to our discussion.

1. An identifier must first be generated. If the identifier is intended to be unique, the generation process includes some mechanism, such as allocation by a central authority, to help ensure uniqueness. However the notion of "unique" involves determining whether a putative identifier matches any other already-allocated identifier. As we will see, for many types of identifiers, this is not simply an exact binary match.

   As a result of generating the identifier, it is often stored in two locations: with the requester or "holder" of the identifier, and with some repository of identifiers (e.g., DNS). For example, if the identifier was allocated by a central authority, the repository might be that authority. If the identifier identifies a device or content on a device, the repository might be that device.
2. The identifier must be distributed, either by the holder of the identifier or by a repository of identifiers, to others who could use the identifier. This distribution might be electronic, but sometimes it is via other channels such as voice, business card, billboard, or other form of advertisement. The identifier itself might be distributed directly, or it might be used to generate a portion of another type of identifier that is then distributed. For example, a URI or email address might include a server name, and hence distributing the URI or email address also inherently distributes the server name.
3. The identifier must be used by some party. Generally the user supplies the identifier which is (directly or indirectly) sent to the repository of identifiers. For example, using an email address to send email to the holder of an identifier may result in the email arriving at the holder's email server which has access to the mail stores.

   The repository of identifiers must then attempt to match the user-supplied identifier with an identifier in its repository.

```
                            +------------+
                            |  Holder of |     1. Generation
                            | identifier +<---------+
                            +----+-------+          |
                                 |                  | Match
                                 |                  v/
                                 |          +-------+-------+
                                 +----------+ Repository of |
                                 |          |  identifiers  |
                                 |          +-------+-------+
                 2. Distribution |                  ^\
                                 |                  | Match
                                 v                  |
                       +---------+-------+          |
                       |      User of    |          |
                       |    identifier   +----------+
                       +-----------------+    3. Use
```

Typical Identifier Processes

Figure 1

One key aspect is that the identifier values passed in generation, distribution, and use, may all be different forms. For example, generation might be exchanged in printed form, distribution done via voice, and use done electronically. As such, the match process can be complicated.

Furthermore, in many uses, the relationship between holder, repositories, and users may be more involved. For example, when a hierarchy of web caches exist, each cache is itself a repository of a sort, and the match process is usually intended to be the same as on the origin server.

1.1. Canonicalization

Perhaps the most common algorithm for comparison involves first converting each identifier to a canonical form (a process known as "canonicalization" or "normalization"), and then testing . the resulting canonical representations for bitwise equality. In so doing, it is thus critical that all entities involved agree on the same canonical form and use the same canonicalization algorithm so that the overall comparison process is also the same.

Note that in some contexts, such as in internationalization, the terms "canonicalization" and "normalization" have a precise meaning. In this document, however, we use these terms synonymously in their more generic form, to mean conversion to some standard form.

While the most common method of comparison includes canonicalization, comparison can also be done by defining an equivalence algorithm, where no single form is canonical. However in most cases, a canonical form is useful for other purposes, such as output, and so in such cases defining a canonical form suffices to define a comparison method.

2. Security Uses

Identifiers such as hostnames, URIs, and email addresses are used in security contexts to identify principals and resources as well as other security parameters such as types and values of claims. Those identifiers are then used to make security decisions based on an identifier supplied via some protocol. For example:

- Authentication: a protocol might match a security principal identifier to look up expected keying material, and then match keying material.
- Authorization: a protocol might match a resource name to look up an access control list (ACL), and then look up the security principal identifier (or a surrogate for it) in that ACL.
- Accounting: a system might create an accounting record for a security principal identifier or resource name, and then might later need to match a supplied identifier to allow (for example) law enforcement to follow up based on the records, or add new filtering rules based on the records in order to stop an attack.

If the parties involved in a security decision use different matching algorithms for the same identifiers, then failure scenarios ranging from denial of service to elevation of privilege can result, as we will see.

This is especially complicated in cases involving multiple parties and multiple protocols. For example, there are many scenarios where some form of "security token service" is used to grant to a requester permission to access a resource, where the resource is held by a third party that relies on the security token service (see Figure 2). The protocol used to request permission (e.g., Kerberos or OAuth) may be different from the protocol used to access the resource (e.g., HTTP). Opportunities for security problems arise when two protocols define different comparison algorithms for the same type of identifier, or when a protocol is ambiguously specified and two endpoints (e.g., a security token service and a resource holder) implement different algorithms within the same protocol.

```
        +----------+
        | security |
        |  token   |
        | service  |
        +----------+
             ^
             | 1. supply credentials and
             | get token for resource
             |                                            +--------+
        +----------+  2. supply token and access resource  |resource|
        |requester |=----------------------------------->| holder |
        +----------+                                      +--------+
```

Simple Security Exchange

Figure 2

In many cases the situation is more complex. With certificates, the name in a certificate gets compared against names in ACLs or other things. In the case of web site security, the name in the certificate gets compared to a portion of the URI that a user may have typed into a browser. The fact that many different people are doing the typing, on many different types of systems, complicates the problem.

Add to this the certificate enrollment step, and the certificate issuance step, and two more parties have an opportunity to adjust the encoding or worse, the software that supports them might make changes that the parties are unaware are happening.

2.1. Types of Identifiers

In this document we will refer to the following types of identifiers:

- Absolute: identifiers that can be compared byte-by-byte for equality. Two identifiers that have different bytes are defined to be different. For example, binary IP addresses are in this class.
- Definite: identifiers that have a well-defined comparison algorithm on which all parties agree. For example, URI scheme names are required to be ASCII and are defined to match in a case-insensitive way; the comparison is thus definite since all parties agree on how to do a case-insensitive match among ASCII strings.
- Indefinite: identifiers that have no single comparison algorithm on which all parties agree. For example, human names are in this class. Everyone might want the comparison to be tailored for their locale, for some definition of locale. In some cases, there may be limited subsets of parties that might be able to agree

(e.g., ASCII users might all agree on a common comparison algorithm whereas users of other Latin scripts, such as Turkish, may not), but identifiers often tend to leak out of such limited environments.

2.2. False Positives and Negatives

It is first worth discussing in more detail the effects of errors in the comparison algorithm. A "false positive" results when two identifiers compare as if they were equal, but in reality refer to two different objects (e.g., security principals or resources). When privilege is granted on a match, a false positive thus results in an elevation of privilege, for example allowing execution of an operation that should not have been permitted otherwise. When privilege is denied on a match (e.g., matching an entry in a block/ deny list or a revocation list), a permissible operation is denied. At best, this can cause worse performance (e.g., a cache miss, or forcing redundant authentication), and at worst can result in a denial of service.

A "false negative" results when two identifiers that in reality refer to the same thing compare as if they were different, and the effects are the reverse of those for false positives. That is, when privilege is granted on a match, the result is at best worse performance and at worst a denial of service; when privilege is denied on a match, elevation of privilege results.

Figure 3 summarizes these effects.

	"Grant on match"	"Deny on match"
False positive	Elevation of privilege	Denial of service
False negative	Denial of service	Elevation of privilege

Effect of False Positives/Negatives

Figure 3

Elevation of privilege is almost always seen as far worse than denial of service. Hence, for URIs for example, Section 6.1 of [RFC3986] states: "comparison methods are designed to minimize false negatives while strictly avoiding false positives".

Thus URIs were defined with a "grant privilege on match" paradigm in mind, where it is critical to prevent elevation of privilege while minimizing denial of service. Using URIs in a "deny privilege on

match" system can thus be problematic.

2.3. Hypothetical Example

In this example, both security principals and resources are identified using URIs. Foo Corp has paid example.com for access to the Stuff service. Foo Corp allows its employees to create accounts on the Stuff service. Alice gets the account "http://example.com/Stuff/FooCorp/alice" and Bob gets "http://example.com/Stuff/FooCorp/bob". It turns out, however, that Foo Corp's URI canonicalizer includes URI fragment components in comparisons whereas example.com's does not, and Foo Corp does not disallow the # character in the account name. So Chuck, who is a malicious employee of Foo Corp, asks to create an account at example.com with the name alice#stuff. Foo Corp's URI logic checks its records for accounts it has created with stuff and sees that there is no account with the name alice#stuff. Hence, in its records, it associates the account alice#stuff with Chuck and will only issue tokens good for use with "http://example.com/Stuff/FooCorp/alice#stuff" to Chuck.

Chuck, the attacker, goes to a security token service at Foo Corp and asks for a security token good for "http://example.com/Stuff/FooCorp/alice#stuff". Foo Corp issues the token since Chuck is the legitimate owner (in Foo Corp's view) of the alice#stuff account. Chuck then submits the security token in a request to "http://example.com/Stuff/FooCorp/alice".

But example.com uses a URI canonicalizer that, for the purposes of checking equality, ignores fragments. So when example.com looks in the security token to see if the requester has permission from Foo Corp to access the given account it successfully matches the URI in the security token, "http://example.com/Stuff/FooCorp/alice#stuff", with the requested resource name "http://example.com/Stuff/FooCorp/alice".

Leveraging the inconsistencies in the canonicalizers used by Foo Corp and example.com, Chuck is able to successfully launch an elevation of privilege attack and access Alice's resource.

Furthermore, consider an attacker using a similar corporation such as "foocorp" (or any variation containing a non-ASCII character that some humans might expect to represent the same corporation). If the resource holder treats them as different, but the security token service treats them as the same, then again elevation of privilege can occur.

3. Common Identifiers

In this section, we walk through a number of common types of identifiers and discuss various issues related to comparison that may affect security whenever they are used to identify security principals or resources. These examples illustrate common patterns that may arise with other types of identifiers.

3.1. Hostnames

Hostnames (composed of dot-separated labels) are commonly used either directly as identifiers, or as components in identifiers such as in URIs and email addresses. Another example is in [RFC5280], sections 7.2 and 7.3 (and updated in section 3 of [I-D.ietf-pkix-rfc5280-clarifications]), which specify use in certificates.

In this section we discuss a number of issues in comparing strings that appear to be some form of hostname.

It is first worth pointing out that the term itself is often ambiguous, and hence it is important that any use clarify which definition is intended. Some examples of definitions include:

a. A Fully-Qualified Domain Name (FQDN),

b. A FQDN that is associated with address records,

c. The leftmost label in an FQDN, or

d. The leftmost label in an FQDN that is associated with address records.

The use of different definitions in different places results in questions such as whether "example" and "example.com" are considered equal or not.

Section 3 of [RFC6055] discusses the differences between a "hostname" vs. a "DNS name", where the former is a subset of the latter by using a restricted set of characters. If one canonicalizer uses the "DNS name" definition whereas another uses a "hostname" definition, a name might be valid in the former but invalid in the latter. As long as invalid identifiers are denied privilege, this difference will not result in elevation of privilege.

[IAB1123] briefly discusses issues with the ambiguity around whether a label will be "alphabetic", including among other issues, how "alphabetic" should be interpreted in an internationalized environment, and whether a hostname can be interpreted as an IP address. We explore this last issue in more detail below.

3.1.1. IPv4 Literals

[RFC1123] section 2.1 states:

> Whenever a user inputs the identity of an Internet host, it SHOULD be possible to enter either (1) a host domain name or (2) an IP address in dotted-decimal ("#.#.#.#") form. The host SHOULD check the string syntactically for a dotted-decimal number before looking it up in the Domain Name System.

and

> This last requirement is not intended to specify the complete syntactic form for entering a dotted-decimal host number; that is considered to be a user-interface issue.

In specifying the inet_addr() API, the POSIX standard [IEEE-1003.1] defines "IPv4 dotted decimal notation" as allowing not only strings of the form "10.0.1.2", but also allows octal and hexadecimal, and addresses with less than four parts. For example, "10.0.258", "0xA000001", and "012.0x102" all represent the same IPv4 address in standard "IPv4 dotted decimal" notation. We will refer to this as the "loose" syntax of an IPv4 address literal.

In section 6.1 of [RFC3493] getaddrinfo() is defined to support the same (loose) syntax as inet_addr():

> If the specified address family is AF_INET or AF_UNSPEC, address strings using Internet standard dot notation as specified in inet_addr() are valid.

In contrast, section 6.3 of the same RFC states, specifying inet_pton():

> If the af argument of inet_pton() is AF_INET, the src string shall be in the standard IPv4 dotted-decimal form: ddd.ddd.ddd.ddd where "ddd" is a one to three digit decimal number between 0 and 255. The inet_pton() function does not accept other formats (such as the octal numbers, hexadecimal numbers, and fewer than four numbers that inet_addr() accepts).

As shown above, inet_pton() uses what we will refer to as the "strict" form of an IPv4 address literal. Some platforms also use the strict form with getaddrinfo() when the AI_NUMERICHOST flag is passed to it.

Both the strict and loose forms are standard forms, and hence a protocol specification is still ambiguous if it simply defines a

string to be in the "standard IPv4 dotted decimal form". And, as a result of these differences, names such as "10.11.12" are ambiguous as to whether they are an IP address or a hostname, and even "10.11.12.13" can be ambiguous because of the "SHOULD" in RFC 1123 above making it optional whether to treat it as an address or a name.

Protocols and data formats that can use addresses in string form for security purposes need to resolve these ambiguities. For example, for the host component of URIs, section 3.2.2 of [RFC3986] resolves the first ambiguity by only allowing the strict form, and the second ambiguity by specifying that it is considered an IPv4 address literal. New protocols and data formats should similarly consider using the strict form rather than the loose form in order to better match user expectations.

If one canonicalizer uses the "loose" definition whereas another uses the "strict" definition, a string might be valid in the former but invalid in the latter. As long as invalid identifiers are denied privilege, this difference will not result in elevation of privilege. Security problems can arise, however, in protocols using strings that conflate IP address literals and hostnames; such strings are at best Definite identifiers, and often turn out to be Indefinite identifiers. (See Section 4 for more discussion.)

Furthermore, when strings can contain non-ASCII characters, they can contain other characters that may look like dots or digits to a human viewing and/or entering the identifier, especially to one who might expect digits to appear in his or her native script.

3.1.2. IPv6 Literals

IPv6 addresses similarly have a wide variety of alternate but semantically identical string representations, as defined in section 2.2 of [RFC4291] and section 2 of [I-D.ietf-6man-uri-zoneid]. As discussed in section 3.2.5 of [RFC5952], this fact causes problems in security contexts if comparison (such as in X.509 certificates), is done between strings rather than between the binary representations of addresses.

[RFC5952] recently specified a recommended canonical string format as an attempt to solve this problem, but it may not be ubiquitously supported at present. And, when strings can contain non-ASCII characters, the same issues (and more, since hexadecimal and colons are allowed) arise as with IPv4 literals.

Whereas (binary) IPv6 addresses are Absolute identifiers, IPv6 address literals are Definite identifiers, since string-to-address conversion for IPv6 address literals is unambiguous.

3.1.3. Internationalization

The IETF policy on character sets and languages [RFC2277] requires support for UTF-8 in protocols, and as a result many protocols now do support non-ASCII characters. When a hostname is sent in a UTF-8 field, there are a number of ways it may be encoded. For example, hostname labels might be encoded directly in UTF-8, or might first be Punycode-encoded [RFC3492] or even percent-encoded from UTF-8.

For example, in URIs, [RFC3986] section 3.2.2 specifically allows for the use of percent-encoded UTF-8 characters in the hostname, as well as the use of IDNA encoding [RFC3490] using the Punycode algorithm.

Percent-encoding is unambiguous for hostnames since the percent character cannot appear in the strict definition of a "hostname", though it can appear in a DNS name.

Punycode-encoded labels (or "A-labels") on the other hand can be ambiguous if hosts are actually allowed to be named with a name starting with "xn--", and false positives can result. While this may be extremely unlikely for normal scenarios, it nevertheless provides a possible vector for an attacker.

A hostname comparator thus needs to decide whether a Punycode-encoded label should or should not be considered a valid hostname label, and if so, then whether it should match a label encoded in some other form such as a percent-encoded Unicode label (U-label).

For example, Section 3 of "Transport Layer Security (TLS) Extensions" [RFC6066], states:

> "HostName" contains the fully qualified DNS hostname of the server, as understood by the client. The hostname is represented as a byte string using ASCII encoding without a trailing dot. This allows the support of internationalized domain names through the use of A-labels defined in [RFC5890]. DNS hostnames are case-insensitive. The algorithm to compare hostnames is described in [RFC5890], Section 2.3.2.4.

For some additional discussion of security issues that arise with internationalization, see [TR36].

3.1.4. Resolution for comparison

Some systems (specifically Java URLs [JAVAURL]) use the rule that if two hostnames resolve to the same IP address(es) then the hostnames are considered equal. That is, the canonicalization algorithm involves name resolution with an IP address being the canonical form.

For example, if resolution was done via DNS, and DNS contained:

```
example.com.  IN A 10.0.0.6
example.net.  CNAME example.com.
example.org.  IN A 10.0.0.6
```

then the algorithm might treat all three names as equal, even though the third name might refer to a different entity.

With the introduction of dynamic IP addresses, private IP addresses, multiple IP addresses per name, multiple address families (e.g., IPv4 vs. IPv6), devices that roam to new locations, commonly deployed DNS tricks that result in the answer depending on factors such as the requester's location and the load on the server whose address is returned, etc., this method of comparison cannot be relied upon. There is no guarantee that two names for the same host will resolve the name to the same IP addresses, nor that the addresses resolved refer to the same entity such as when the names resolve to private IP addresses, nor even that the system has connectivity (and the willingness to wait for the delay) to resolve names at the time the answer is needed.

In addition, a comparison mechanism that relies on the ability to resolve identifiers such as hostnames to other identifies such as IP addresses leaks information about security decisions to outsiders if these queries are publicly observable.

Finally, it is worth noting that determine whether two (unequal) identifiers refer to the same entity is a function of the use of such identifiers, not about actually comparing the identifiers, which is the focus of this document.

3.2. Ports and Service Names

Port numbers and service names are discussed in depth in [RFC6335]. Historically, there were port numbers, service names used in SRV records, and mnemonic identifiers for assigned port numbers (known as port "keywords" at [IANA-PORT]). The latter two are now unified, and various protocols use one or more of these types in strings. For example, the common syntax used by many URI schemes allows port numbers but not service names. Some implementations of the getaddrinfo() API support strings that can be either port numbers or port keywords (but not service names).

For protocols that use service names that must be resolved, the issues are the same as those for resolution of addresses in Section 3.1.4. In addition, Section 5.1 of [RFC6335] clarifies that service names/port keywords must contain at least one letter. This

prevents confusion with port numbers in strings where both are allowed.

3.3. URIs

This section looks at issues related to using URIs for security purposes. For example, [RFC5280], section 7.4, specifies comparison of URIs in certificates. Examples of URIs in security token-based access control systems include WS-*, SAML-P and OAuth WRAP. In such systems, a variety of participants in the security infrastructure are identified by URIs. For example, requesters of security tokens are sometimes identified with URIs. The issuers of security tokens and the relying parties who are intended to consume security tokens are frequently identified by URIs. Claims in security tokens often have their types defined using URIs and the values of the claims can also be URIs.

Also, when a URI is embedded in plain text (e.g., an email message), there is an additional concern because there is no termination criterion for a URI. For example, consider http://unicode.org/cldr/utility/list-unicodeset.jsp?a=a&amp;g=gc. Some applications that detect URIs will stop before the first '.' in the path, while others go to last '.', and yet others may stop at the ';'. As another point of comparison, Section 2.37 of [EE] (a standard for history citations) specifies the use of a space after a URI and before the punctuation.

URIs are defined with multiple components, each of which has its own rules. We cover each in turn below. However, it is also important to note that there exist multiple comparison algorithms. [RFC3986] section 6.2 states:

> A variety of methods are used in practice to test URI equivalence. These methods fall into a range, distinguished by the amount of processing required and the degree to which the probability of false negatives is reduced. As noted above, false negatives cannot be eliminated. In practice, their probability can be reduced, but this reduction requires more processing and is not cost-effective for all applications.
> If this range of comparison practices is considered as a ladder, the following discussion will climb the ladder, starting with practices that are cheap but have a relatively higher chance of producing false negatives, and proceeding to those that have higher computational cost and lower risk of false negatives.

The ladder approach has both pros and cons. On the pro side, it allows some uses to optimize for security, and other uses to optimize for cost, thus allowing URIs to be applicable to a wide range of

uses. A disadvantage is that when different approaches are taken by different components in the same system using the same identifiers, the inconsistencies can result in security issues.

3.3.1. Scheme component

[RFC3986] defines URI schemes as being case-insensitive ASCII and in section 6.2.2.1 specifies that scheme names should be normalized to lower-case characters.

New schemes can be defined over time. In general two URIs with an unrecognized scheme cannot be safely compared, however. This is because the canonicalization and comparison rules for the other components may vary by scheme. For example, a new URI scheme might have a default port of X, and without that knowledge, a comparison algorithm cannot know whether "example.com" and "example.com:X" should be considered to match in the authority component. Hence for security purposes, it is safest for unrecognized schemes to be treated as invalid identifiers. However, if the URIs are only used with a "grant access on match" paradigm then unrecognized schemes can be supported by doing a generic case-sensitive comparison, at the expense of some false negatives.

3.3.2. Authority component

The authority component is scheme-specific, but many schemes follow a common syntax that allows for userinfo, host, and port.

3.3.2.1. Host

Section 3.1 discussed issues with hostnames in general. In addition, [RFC3986] section 3.2.2 allows future changes using the IPvFuture production. As with IPv4 and IPv6 literals, IPvFuture formats may have issues with multiple semantically identical string representations, and may also be semantically identical to an IPv4 or IPv6 address. As such, false negatives may be common if IPvFuture is used.

3.3.2.2. Port

See discussion in Section 3.2.

3.3.2.3. Userinfo

[RFC3986] defines the userinfo production that allows arbitrary data about the user of the URI to be placed before '@' signs in URIs. For example: "http://alice:bob:chuck@example.com/bar" has the value "alice:bob:chuck" as its userinfo. When comparing URIs in a security

context, one must decide whether to treat the userinfo as being significant or not. Some URI comparison services for example treat "http://alice:ick@example.com" and "http://example.com" as being equal.

When the userinfo is treated as being significant, it has additional considerations (e.g., whether it is case-sensitive or not) which we cover in Section 3.4.

3.3.3. Path component

[RFC3986] supports the use of path segment values such as "./" or "../" for relative URIs. Strictly speaking, including such path segment values in a fully qualified URI is syntactically illegal but [RFC3986] section 4.1 nevertheless defines an algorithm to remove them.

Unless a scheme states otherwise, the path component is defined to be case-sensitive. However, if the resource is stored and accessed using a filesystem using case-insensitive paths, there will be many paths that refer to the same resource. As such, false negatives can be common in this case.

3.3.4. Query component

There is the question as to whether "http://example.com/foo", "http://example.com/foo?", and "http://example.com/foo?bar" are each considered equal or different.

Similarly, it is unspecified whether the order of values matters. For example, should "http://example.com/blah?ick=bick&foo=bar" be considered equal to "http://example.com/blah?foo=bar&ick=bick"? And if a domain name is permitted to appear in a query component (e.g., in a reference to another URI), the same issues in Section 3.1 apply.

3.3.5. Fragment component

Some URI formats include fragment identifiers. These are typically handles to locations within a resource and are used for local reference. A classic example is the use of fragments in HTTP URIs where a URI of the form "http://example.com/blah.html#ick" means retrieve the resource "http://example.com/blah.html" and, once it has arrived locally, find the HTML anchor named ick and display that.

So, for example, when a user clicks on the link "http://example.com/blah.html#baz" a browser will check its cache by doing a URI comparison for "http://example.com/blah.html" and, if the resource is present in the cache, a match is declared.

Hence comparisons for security purposes typically ignore the fragment component and treat all fragments as equal to the full resource. However, if one were actually trying to compare the piece of a resource that was identified by the fragment identifier, ignoring it would result in potential false positives. For example, there is at least one well known site today (Twitter) that requires the fragment component in order to uniquely identify a user profile.

3.3.6. Resolution for comparison

As with Section 3.1.4 for hostnames, it may be tempting to define a URI comparison algorithm based on whether they resolve to the same content. Similar problems exist, however, including content that dynamically changes over time or based on factors such as the requester's location, potential lack of external connectivity at the time/place comparison is done, potentially undesirable delay introduced, etc.

In addition, as noted in Section 3.1.4, resolution leaks information about security decisions to outsiders if the queries are publicly observable.

3.4. Email Address-like Identifiers

Section 3.4.1 of [RFC5322] defines the syntax of an email address-like identifier, and Section 3.2 of [RFC6532] updates it to support internationalization. [RFC5280], section 7.5, further discusses the use of internationalized email addresses in certificates.

[RFC6532] use in certificates points to [RFC6530], where Section 13 of that document contains a discussion of many issues resulting from internationalization.

Email address-like identifiers have a local part and a domain part. The issues with the domain part are essentially the same as with hostnames, covered earlier.

The local part is left for each domain to define. People quite commonly use email addresses as usernames with web sites such as banks or shopping sites, but the site doesn't know whether foo@example.com is the same person as FOO@example.com. Thus email-like identifiers are typically Indefinite identifiers.

To avoid false positives, some security mechanisms (such as [RFC5280]) compare the local part using an exact match. Hence, like URIs, email address-like identifiers are designed for use in grant-on-match security schemes, not in deny-on-match schemes.

4. General Conflation Issues

There are a number of examples (some in the preceding sections) of strings that conflate two types of identifiers, using some heuristic to try to determine which type of identifier is given. Similarly, two ways of encoding the same type of identifier might be conflated within the same string.

Some examples include:

1. A string that might be an IPv4 address literal or an IPv6 address literal
2. A string that might be an IP address literal or a hostname
3. A string that might be a port number or a service name
4. A DNS label that might be literal or be Punycode-encoded

Strings that allow such conflation can only be considered Definite if there exists a well-defined rule to determine which identifier type is meant. One way to do so is to ensure that the valid syntax for the two is disjoint (e.g., for IPv4 vs IPv6 address literals). A second way to do so is to define a precedence rule that results in some identifiers being inaccessible via a conflated string (e.g., a host literally named "xn--de-jg4avhby1noc0d" may be inaccessible due to the "xn--" prefix denoting the use of Punycode encoding). In some cases, such inaccessible space may be reserved so that the actual set of identifiers in use are unambiguous. For example, Section 2.5.5.2 of [RFC4291] defines a range of the IPv6 address space for representing IPv4 addresses.

5. General Internationalization Issues

In addition to the issues with hostnames discussed in Section 3.1.3, there are a number of internationalization issues that apply to many types of Definite and Indefinite identifiers.

First, there is no DNS mechanism for identifying whether non-identical strings would be seen by a human as being equivalent. There are problematic examples even with ASCII (Basic Latin) strings including regional spelling variations such as "color" and "colour" and many non-English cases including partially-numeric strings in Arabic script contexts, Chinese strings in Simplified and Traditional forms, and so on. Attempts to produce such alternate forms algorithmically could produce false positives and hence have an adverse affect on security.

Second, some strings are visually confusable with others, and hence if a security decision is made by a user based on visual inspection, many opportunities for false positives exist. As such, using visual

inspection for security is unreliable. In addition to the security issues, visual confusability also adversely affects the usability of identifiers distributed via visual mediums. Similar issues can arise with audible confusability when using audio (e.g., for radio distribution, accessibility to the blind, etc.) in place of a visual medium.

Determining whether a string is a valid identifier should typically be done after, or as part of, canonicalization. Otherwise an attacker might use the canonicalization algorithm to inject (e.g., via percent encoding, NFKC, or non-shortest-form UTF-8) delimiters such as '@' in an email address-like identifier, or a '.' in a hostname.

Any case-insensitive comparisons need to define how comparison is done, since such comparisons may vary by locale of the endpoint. As such, using case-insensitive comparisons in general often result in identifiers being either Indefinite or, if the legal character set is restricted (e.g., to ASCII), then Definite.

See also [WEBER] for a more visual discussion of many of these issues.

Finally, the set of permitted characters and the canonical form of the characters (and hence the canonicalization algorithm) sometimes varies by protocol today, even when the intent is to use the same identifier, such as when one protocol passes identifiers to the other. See [I-D.ietf-precis-problem-statement] for further discussion.

6. General Scope Issues

Another issue arises when an identifier (e.g., "localhost", "10.11.12.13", etc.) is not globally unique. [RFC3986] Section 1.1 states:

> URIs have a global scope and are interpreted consistently regardless of context, though the result of that interpretation may be in relation to the end-user's context. For example, "http://localhost/" has the same interpretation for every user of that reference, even though the network interface corresponding to "localhost" may be different for each end-user: interpretation is independent of access.

Whenever a non-globally-unique identifier is passed to another entity outside of the scope of uniqueness, it will refer to a different resource, and can result in a false positive. This problem is often

addressed by using the identifier together with some other unique identifier of the context. For example "alice" may uniquely identify a user within a system, but must be used with "example.com" (as in "alice@example.com") to uniquely identify the context outside of that system.

It is also worth noting that non-globally-scoped IPv6 addresses can be written with, or otherwise associated with, a "zone ID" to identify the context (see [RFC4007] for more information). However, zone IDs are only unique within a host, so they typically narrow, rather than expand, the scope of uniqueness of the resulting identifier.

7. Security Considerations

This entire document is about security considerations.

To minimize elevation of privilege issues, any system that requires the ability to use both deny and allow operations within the same identifier space, should avoid the use of Indefinite identifiers in security comparisons.

To minimize future security risks, any new identifiers being designed should specify an Absolute or Definite comparison algorithm, and if extensibility is allowed (e.g., as new schemes in URIs allow) then the comparison algorithm should remain invariant so that unrecognized extensions can be compared. That is, security risks can be reduced by specifying the comparison algorithm, making sure to resolve any ambiguities pointed out in this document (e.g., "standard dotted decimal").

Some issues (such as unrecognized extensions) can be mitigated by treating such identifiers as invalid. Validity checking of identifiers is further discussed in [RFC3696].

Perhaps the hardest issues arise when multiple protocols are used together, such as in the figure in Section 2, where the two protocols are defined or implemented using different comparison algorithms. When constructing an architecture that uses multiple such protocols, designers should pay attention to any differences in comparison algorithms among the protocols, in order to fully understand the security risks. An area for future work is how to deal with such security risks in current systems.

8. Acknowledgements

Yaron Goland contributed to the discussion on URIs. Patrik Faltstrom contributed to the background on identifiers. John Klensin contributed text in a number of different sections. Additional helpful feedback and suggestions came from Bernard Aboba, Leslie Daigle, Mark Davis, Russ Housley, Magnus Nystrom, and Chris Weber.

9. IANA Considerations

This document requires no actions by the IANA.

10. Informative References

[EE] Mills, E., "Evidence Explained: Citing History Sources from Artifacts to Cyberspace", 2007.

[I-D.ietf-6man-uri-zoneid]
Carpenter, B. and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", draft-ietf-6man-uri-zoneid-02 (work in progress), July 2012.

[I-D.ietf-pkix-rfc5280-clarifications]
Yee, P., "Updates to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", draft-ietf-pkix-rfc5280-clarifications-08 (work in progress), August 2012.

[I-D.ietf-precis-problem-statement]
Blanchet, M. and A. Sullivan, "Stringprep Revision and PRECIS Problem Statement", draft-ietf-precis-problem-statement-07 (work in progress), August 2012.

[IAB1123] IAB, "The interpretation of rules in the ICANN gTLD Applicant Guidebook", February 2012, <http://www.iab.org/documents/correspondence-reports-documents/2012-2/iab-statement-the-interpretation-of-rules-in-the-icann-gtld-applicant-guidebook>.

[IANA-PORT]
IANA, "PORT NUMBERS", June 2011, <http://www.iana.org/assignments/port-numbers>.

[IEEE-1003.1]

IEEE and The Open Group, "The Open Group Base Specifications, Issue 6 IEEE Std 1003.1, 2004 Edition", IEEE Std 1003.1, 2004.

[JAVAURL] Oracle, "Class URL, Java(TM) Platform, Standard Ed. 7", 2011, <http://docs.oracle.com/javase/7/docs/api/java/net/URL.html>.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

[RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, March 2003.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.

[RFC3696] Klensin, J., "Application Techniques for Checking and Transformation of Names", RFC 3696, February 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4007] Deering, S., Haberman, B., Jinmei, T., Nordmark, E., and B. Zill, "IPv6 Scoped Address Architecture", RFC 4007, March 2005.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, October 2008.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.

[RFC6055] Thaler, D., Klensin, J., and S. Cheshire, "IAB Thoughts on Encodings for Internationalized Domain Names", RFC 6055, February 2011.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.

[RFC6530] Klensin, J. and Y. Ko, "Overview and Framework for Internationalized Email", RFC 6530, February 2012.

[RFC6532] Yang, A., Steele, S., and N. Freed, "Internationalized Email Headers", RFC 6532, February 2012.

[TR36] Unicode Consortium, "Unicode Security Considerations", Unicode Technical Report 36, August 2004.

[WEBER] Weber, C., "Attacking Software Globalization", March 2010, <http://www.lookout.net/files/Chris_Weber_Character%20Transformations%20v1.7_IUC33.pdf>.

Author's Address

Dave Thaler (editor)
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA

Phone: +1 425 703 8835
Email: dthaler@microsoft.com