Pathfinder—A new approach to Trust Management

Patrick Patterson
Carillon Information Security Inc.
ppatterson@carillon.ca

Dave Coombs
Carillon Information Security Inc.
dcoombs@carillon.ca

Abstract

PKI has long promised to solve the problem of scalable identity management for us. Until now, that promise has been rather empty, especially in the Free and Open-Source Software (FOSS) space. Generally speaking, the problem is that making the proper trust decisions when presented with a certificate involves such esoteric magic as checking CRLs and OCSP and validating trust and policy chains, all of which are expressed as arcane ASN.1 structures; usually this is not the application developer's primary focus. Coupled with the lack of central PKI management tools in the FOSS environment, the result is a whole lot of applications that sort of do PKI, but not in any truly useful fashion. That is, an administrator cannot really replace username/password systems with certificates, which is what PKI was supposed to let us do. Microsoft has finally built a decent certificate-handling framework into their products, and we believe that Pathfinder adds this level of support to FOSS products. This presentation will focus on what Pathfinder is, how it can be used to deploy a scalable trust management framework, and, most importantly, will demonstrate how easy it is to make your own application "Pathfinder aware."

1 Introduction

Before we discuss Pathfinder in detail, it is useful to take a look at Public Key Infrastructure in general, its history, and the current directions within this field. This provides a backdrop and common reference for understanding why we have chosen the architecture for trust management with Pathfinder that we have.

While PKI is, generally speaking, not new technology, it has been quite slow to find mainstream adoption and use. This is partly because of the difficulty in designing an implementation that satisfies everybody's requirements, and partly because of interoperability problems arising from a range of implementations that interpret the standards differently. Recent developments, however, show great promise in growing interoperable deployments.

Formerly it was thought that the "holy grail" of PKI was a single, global Certificate Authority, or small group of such CAs, that would issue all the certificates, and that everybody would trust these CAs. This would eventually prove to be impossible, and the biggest problem is a political one: who runs the global CA? Who is, therefore, the global trust authority, and why should one group have so much power? These questions could not be satisfactorily answered. Furthermore, many different groups and interests are represented in such a system, and it is probably impossible to create a meaningful global PKI policy framework that is consistent with the needs of these different groups and different industries and different legal regulations.

In the past several years a new model has emerged, represented by interoperable communities of trust. Groups of people or companies with similar requirements, be they industry requirements, legal requirements or otherwise, can define a community of trust and a set of policies and procedures that are suitable for that community, which then can be adopted and followed by all participants.

The current way forward for these communities of trust is to use Bridge Certificate Authorities. A Bridge CA is one that doesn't issue certificates to subscribers itself, but rather exists to facilitate the trust fabric within a community. If a bridge CA is set up inside a specific community, participating community members with their own CAs can cross-certify with the bridge, using policy mapping to create equivalence among assurance levels which can allow trust to flow through the community. There are two principal benefits of using a bridge. The first is that participant CAs do not have to cross-certify with every other CA in the community, and instead only manage a single audited trust relationship.

The second benefit is that a user can declare an identity to the community, and all the participants in the community can recognize that identity. The bridge CA can also then cross-certify with other bridges, allowing trust to flow to other communities as well.

Figure 1 shows the current status of certain interconnected communities of trust that already exist. Currently the US Federal Bridge CA is acting as a "super bridge," cross-certified with government departments, but also cross-certified with an aerospace and defence bridge, a pharmaceutical bridge, and a higher education bridge. It is to be noted that each of these communities is standalone, and each of the companies listed operates its own PKI. The arrows only represent policy mapping between each of the participants.

Pathfinder was conceived as a method to allow FOSS projects to seamlessly handle the complexities inherent in this cross-certified "community PKI" framework.

2 Technical Expressions of Trust

IETF RFC3280 describes a detailed, standard profile for expressing an identity in a PKI context, and methods for ascertaining the status and current validity of such an identity. X.509 certificates have a Subject field, which can hold some representation of "identity," and the RFC3280 standard Internet profile also includes Subject Alternative Names that can express email addresses, DNS names, any many other forms (some people think that too many things can be expressed here!) There is also the Authority Information Access extension, which can contain an LDAP or HTTP pointer to download the signing certificate of the CA that issued and signed a given certificate, as well as a pointer to the Online Certificate Status Protocol (OCSP) service that can give revocation status about this certificate. If the CA doesn't support OCSP, a relying party can fall back to checking the Certificate Revocation List (CRL), a pointer to which can be found in the certificate's CRL Distribution Points extension. We can check the policy under which a certificate was issued, as found in the Certificate Policies extension. In a bridged environment, we may encounter Policy Mapping, so we will have to check that extension as well. And we mustn't forget Name Constraints, which, within a bridged (or even a hierarchical) PKI allow delegation of authority over name spaces such as email, DNS, and X.500, to particular Certificate Authorities.

Given the above, we can identify several hurdles. First of all, performing certificate validation correctly pretty much requires one to be a PKI expert, as there is a high degree of complexity involved. We shouldn't necessarily expect application programmers, who are experts in building web servers, mail servers, RADIUS servers, or other applications where using certificates for authentication may be desirable, to stop and learn how to do this validation correctly. It probably isn't their primary concern. Secondly, we shouldn't necessarily expect libraries commonly used for functions like TLS to help us out of this, at least not fully. For instance, OpenSSL and libNSS are very good security libraries, but there are just too many details involved in building a trust path to expect these libraries to possess all of the required capability. As a specific example, we rather strongly doubt that the OpenSSL maintainers will ever want to integrate an HTTP and LDAP client in their library and provide all the hooks necessary to synchronously and/or asynchronously fetch certificates, CRLs, and OCSP information from Certificate Authorities.

2.1 So, where does this leave us?

The situation today is that most applications that implement any form of certificate support do a rather poor job of it. Most commonly, they implement the capability to act as the server portion of a TLS connection, and either stop there or offer very rudimentary (in our experience) client authentication support, often limited to checking whether the certificate presented is signed by a known and trusted CA and is in its validity period. The need to somehow check certificate status, for example by CRL or OCSP, is mostly ignored, as is the requirement in some communities to only accept certificates issued to a certain policy. There is certainly no thought given to how to deploy such an application in an environment with a complex trust fabric, such as the above cross-certified domains, where the aforementioned name constraints and policy mappings need to be evaluated. Furthermore, client side validation of server certificates is almost never properly implemented, which is understandable in that it is very difficult to communicate to an end user what to do when, for example, the server's certificate is revoked.

To solve these issues, it is unrealistic to expect each application developer to be a PKI expert and correctly implement all the complexity and nuance of the Path Dis-

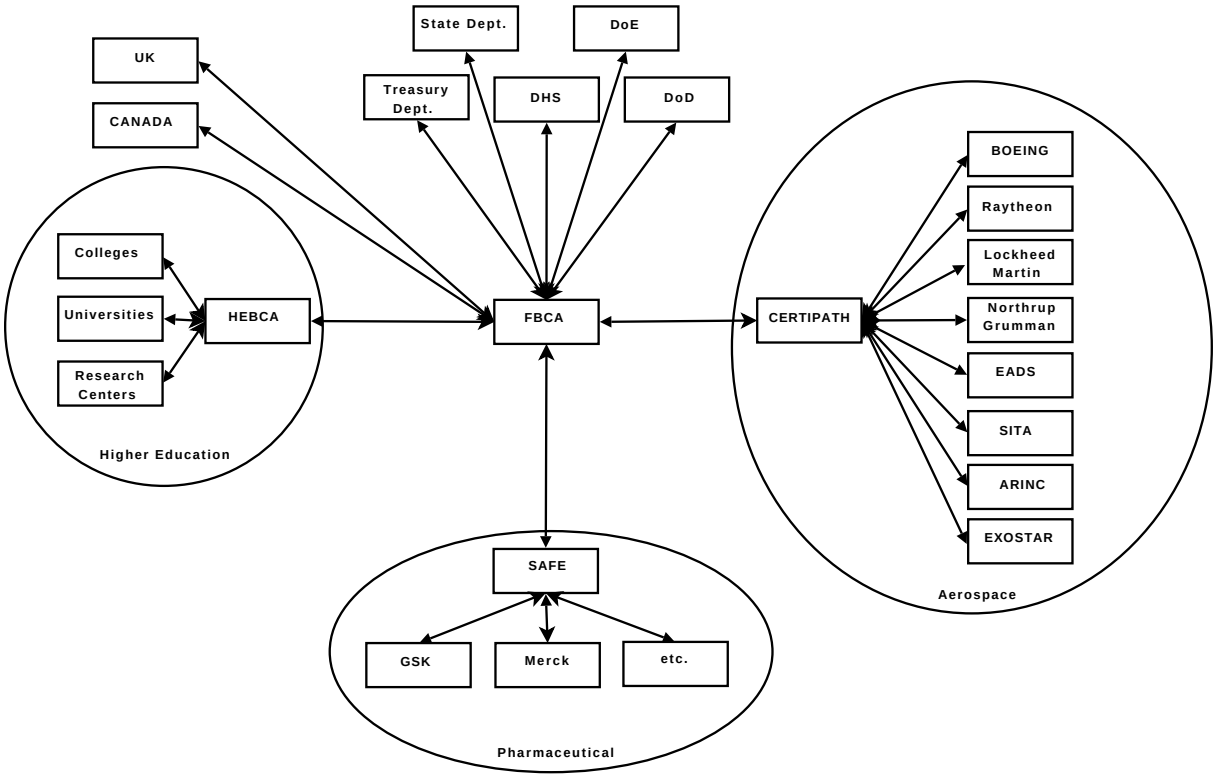

Figure 1: Interconnected Communities of trust, present and planned

covery and Validation algorithm described in RFC 3280. Therefore, another solution is needed.

Pathfinder solves the above problem by providing two components. The first is a series of client libraries that provide callbacks for certificate validation for all major security libraries (currently OpenSSL and libNSS, but hooks to the Java CryptoAPI and GNUTLS are also planned), and which imposes on applications only one additional dependency: the need to link with DBus, which the client uses to talk to the daemon

The second component is a system daemon that provides all of the certificate validation functions in a transparent and centrally manageable fashion. This approach allows an application to grow the capability of handling a complex trust environment simply, usually by only replacing a single line of code, and perhaps by adding an option for the policy to be set for that application by its configuration. All of the hard work is done in the daemon.

3 Advantages of Pathfinder

Pathfinder was written with the credo "Do the hard stuff once, let everyone benefit" in mind.

In a complex trust environment such as a bridged PKI, it is critical that any Policy Mapping and Name Constraints extensions are correctly handled. As illustrated by Figure 2, Policy Mapping enables a company to declare that its policy for issuing certificates is equivalent to another company's policy, often via a central, community-endorsed policy (that of the bridge.) Differentiation by policy, when evaluating a given certificate, is also useful as it is now fairly common not to include policy information in the Distinguished Name, including it instead only where it belongs, in the Certificate Policies extension. So, it would be possible to have two certificates with the same Distinguished Name (they refer to the same security principal after all), but issued according to a different policy. Without a way to tell an application only to accept certificates issued according to a certain policy, we have a security issue if the user presents a certificate issued according to a less stringent policy.

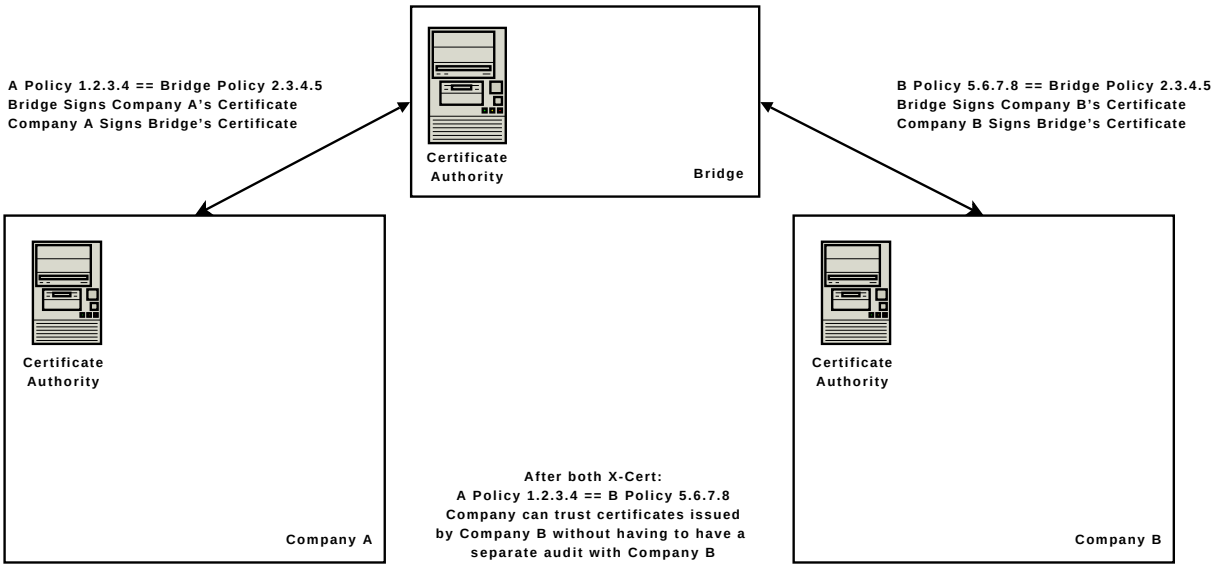

Figure 2: Policy Mapping in a bridge environment

Name constraints are required to ensure that only a single organisation in the trust fabric can be authoritative for a particular instance of a security principal (in plain terms, you usually only want John Doe who works at Company A to have a certificate issued by Company A.) Another way to put this is that name constraints can be used to state that a given CA can only issue certificates to Subjects inside a given namespace (for instance, email domain), and can be used to state that no other CA, for example across the bridge, can issue certificates in that namespace.

The procedures for correctly handling policy mapping and name constraints run to several dozens of pages within various RFCs and specifications, and it is certainly a sufficiently tricky task that one would hope it is only done once on a given platform. As more and more applications are being deployed into the, rather large, communities mentioned in the introduction, it is becoming more and more desirable for applications to all handle these complex issues, and to handle them in a consistent manner.

It is equally important that the status of a certificate be validated, to ensure that the certificate has not been revoked. And, perhaps most interesting for those deploying an application in a complex trust environment, ?missing? certificates must be obtained, and a trust path between the client certificate and the configured trust anchor must be built. These transactions may involve round trips to external LDAP, HTTP, and OCSP servers, and thus, unless done very efficiently, may induce transaction delays and substantially impact server performance.

Which leads us to the question of providing a scalable caching layer.

When performing Path Discovery and Validation, there are several potential bottlenecks. Clearing these bottlenecks is highly desirable, since performing the certificate validation is a blocking function in the establishment of protocols such as IPsec and TLS. That is, there is no opportunity to send the request in an asynchronous manner. A Certificate is presented, and it must be validated there, on the spot, before proceeding with the rest of the establishment of the session. The functions where caching is most desirable are Authority Information Access (AIA) chasing, CRL downloading, and making an OCSP request. For all of these we have to worry about, from a performance point of view, the latency of performing the query, and the DNS or TCP timeouts if the host listed is not available. For CRLs, we have the additional problem that what we download is of an unknown, arbitrary size, and may be quite large (for instance, the US Department of Defence CRL was over 50MB at one point).

Optimization of the above functions is perhaps one of the greatest reasons for choosing a centralized daemon model for Pathfinder. By performing everything in the daemon, we can optimise these functions in a single lo-

```
Apache:

Before:
Incomplete support for complex trust environments:
No support for Policy Mapping
Must supply all intermediate certificates, which is undesirable in a
properly configured bridge environment.
No support for Name Constraints, as a matter of fact, if they are present
and critical, the certificate will not validate at all, which is correct
behaviour, but it means that Apache can't be deployed using Certificate
based authentication in much of the aerospace industry).
No support for OCSP
No support for certificate validation based on Certificate Policy
CRL update requires re-starting the server.
Per server configuration of trust anchors

After:
Support for full RFC3280 Path Discovery and Validation
Full support for CRL and OCSP.
Specification of policy against which a certificate would be validated.
Full support for policy mapping.
All servers within a farm may be configured at once, and trust anchors
updated to all.

Time to implement:
2 days

Biggest Challenge:
Dealing with the apache configure/makefile system.

Additional Dependencies:
libdbus
```

Figure 3: A concrete example

cation to minimise a very critical performance bottleneck, and we also have the greatest opportunity to re-use information already obtained. For instance, if application A requests the certificate chain W->X -> Y -> Z , and we need to fetch X and Y, then we can cache those certificates for the duration of their lifetimes, and can therefore avoid having to re-fetch the same certificates, thus saving time for application B that requests the same chain at some future date. Changes to the chain can be managed by having a maintenance thread which periodically checks the chain for any changes, such as refreshed certificates with different CRL or OCSP URIs, or different lifetimes. The same can be done with the CRL and OCSP responses, although, of course, the maintenance thread will have to refresh more frequently, due to the shorter lifetimes of these artifacts.

Performing these functions in one location also offers the option of a graceful "offline" mode, in case the validation is performed offline, such as a user validating an S/MIME signature while on a flight. Instead of each application needing built-in logic for permissible failure modes, Pathfinder can be centrally configured to perform the appropriate level of validation, thereby ensuring that each application handles offline validation in a consistent manner. An example of such a validation scheme is to have three settings, the first of which is to require full validation, which will fail to validate the signature because it can't find the revocation information when offline, and may not be able to chase AIA information. The second setting is to require a valid trust chain but not necessarily fresh revocation info, which may work, depending on how fresh the cache is, and

the the third setting is to blindly accept all certificates if offline, which will always validate, assuming that the certificate hasn't expired.

This last feature is not yet implemented, but is definitely on the roadmap.

4 Conclusion

Proper handling of certain X.509 certificate extensions when performing certificate validation is essential in order to maintain a viable trust fabric in a bridged PKI framework serving a community of trust. In such a community, each participant need only be responsible for one trust relationship: its relationship with the bridge. The bridge, then, brokers trust among the community participants who have agreed to a common policy framework. However, in order for this to work, there is a substantial requirement on the various server software packages used in the community to correctly process the trust path and policy tree. Until now, due to the complexity of handling this processing, it has been difficult for application developers to deploy full PKI support in their applications. Pathfinder not only makes this simple, it provides a scalable and manageable way to deploy true PKI-enabled applications using only open source software.

Proceedings of the Linux Symposium

Volume Two

July 23rd–26th, 2008
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
Matthew Wilson, *rPath*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
Eugene Teo, *Red Hat, Inc.*
Kyle McMartin, *Red Hat, Inc.*
Jake Edge, *LWN.net*
Robyn Bergeron
Dave Boutcher, *IBM*
Mats Wichmann, *Intel*