Project Moonshot: Feasibility Analysis

Painless Security

February 1, 2010

The opinions expressed are those of the authors and may not reflect those of JANET(UK)

Table of Contents

1. Executive Summary 3
2. Introduction 4
2.1. Goals 5
2.2. Technical Feasibility 5
2.3. Applicability 5
2.4. Credentials 6
2.5. Applications 6
3. Target Architecture 8
4. EAP in the Federated Model 9
4.1. Summary 10
5. GSS-API and Application Authentication 11
5.1. Summary 11
6. EAP and GSS-API Together 12
6.1. Negotiating the EAP Method 12
6.2. Reliability 12
6.3. EAP Channel Binding 13
6.4. Naming and Authorization 14
6.5. Security Layers, GSS-API Channel Binding and PRF 14
6.6. Summary 15
7. Authorization and Personalization 17
8. Metadata Maintenance 18
9. Work Plan 19
9.1. Standardization Work Required 19
9.2. Implementing a Proof-of-Concept Demonstration 19
10. Alternatives 22
10.1. Information Card 23
10.2. Raw SAML GSS-API Mechanism 23
10.3. Backend GSS into an Authentication Server 23
10.4. Backend EAP into Kerberos pre-authentication and use IAKERB 24
11. Conclusions 26

1. Executive Summary

Today, there are a number of solutions that provide federated identity management for the web. That is, these solutions allow an organization providing services over the web to consume the identities of its research partners, business partners, customers or other affiliates. The users of these services can use the same credential (such as a user name and password) when they are accessing a service local to their organization that they would use when accessing a partner service. In addition, the same business processes that manage their roles and status within an organization provide partners with information that those partners need in order to provide the service. However, today there is no good solution to take this federated identity management to applications beyond the web such as e-mail, instant messaging, file sharing, or remote computing.

Project Moonshot proposes to combine three key security technologies to build a test-bed for extending the benefits of federated identity management to these applications. The Extensible Authentication Protocol (EAP) is used in the JANET Roaming Service in the UK, and eduroam on a global level, to provide federation for network access. The Generic Security Services Application Programming Interface (GSS-API) provides a way to add security mechanisms to most popular application protocols and to a number of implementations of these protocols; GSS-API technology is at the core of a number of vendor security solutions including Microsoft's Active Directory. The Security Assertion Mark-up Language (SAML) serves to transport information about an identity from an identity provider to a service provider. SAML is a core technology behind the UK Access Management Federation. Moonshot also aims to leverage these security technologies to improve the maintenance of technical information about members of a federation transported in the form of SAML metadata to support the growth of the UK Access Management Federation and similar organizations.

Painless Security has been contracted to provide an independent evaluation of the technical feasibility of Moonshot. This report provides that evaluation. The approach of using EAP, GSS-API and SAML to provide federated identity management is technically sound. As expected when reviewing an early-stage project, a number of issues have been identified that should be closed during the design, standardization and implementation of Moonshot; resolving these issues should not call into question the overall technical feasibility of the project. Discussions of the proposal to simplify SAML metadata maintenance have raised a number of issues and opportunities for simplification and enhancement. While the shape of the metadata proposal is undergoing active revision, the evaluation to date suggests that these revisions will lead to a technically feasible proposal for using Moonshot to simplify metadata maintenance.

Ultimately, technical feasibility is one relatively small part of Moonshot's success. Painless Security believes that the most important factor in Moonshot's success is whether client application authors will be able to take advantage of the technology. For this to be possible, the technology needs to be available in vendor platforms such as Windows and Java, as well as in frameworks used to make client applications. In addition, the technology needs to be easy to use and needs to meet client applications' design requirements. Achieving this will be largely political and social rather than technical. However providing clear examples and convenience frameworks for popular application development environments will also be important.

2. Introduction

Today, there are a number of solutions that provide federated identity management for the web. That is, these solutions allow an organization providing services over the web to consume the identities of its research partners, business partners, customers or other affiliates. The users of these services can use the same credential (such as a user name and password) when they are accessing a service local to their organization that they would use when accessing a partner service. In addition, the same business processes that manage their roles and status within an organization provide partners with information that those partners need in order to provide the service. One common solution to this problem is the Security Assertion Mark-up Language (SAML) Web SSO Profile. In this model, a service provider receives an assertion from an identity provider that conveys information about a user of the service.

However, today there is no good solution to take this federated identity management to applications beyond the web such as e-mail, instant messaging, and file sharing or remote computing. In order to meet the needs of its clients and the broader European academic community, JANET(UK) wishes to develop a test bed for such a solution and foster development of products taking advantage of this solution. At the same time, JANET(UK) would like to address the scalability of existing SAML federations that it manages and participates in. Today, participants in the federation are given "batched metadata," a file containing technical information about the participants in the federation. As the federation grows this approach becomes unwieldy just as the approach of distributing a "hosts file" containing information on all hosts on the Internet grew unwieldy as the Internet grew beyond a few hundred hosts. For both the case of SAML metadata and the hosts file, a small number of consumers can afford to locally store all the necessary information and update this local store on a frequent basis. As the number of consumers and size of information increases, storing a complete copy at each consumer becomes unworkable. Instead, solutions that permit consumers to request the information they need reasonably close to the time it is needed scale better. If federation expands beyond the web, then this new usage will add to the existing growth in metadata. Aspects of a more real-time system for maintaining metadata may directly take advantage of the new application federation technology. Also, if the federation technology is carefully designed it may simplify the requirements for metadata distribution.

JANET(UK) has contracted with Painless Security to produce this report evaluating the technical feasibility of Project Moonshot, the proposal to develop this federated authentication model.

Painless Security provides advice and technical expertise to its clients in the areas of networking, security, infrastructure software and system architecture. Painless Security brings significant experience with application identity management and the specific security technologies used in the Moonshot proposal to the evaluation. Painless Security founder Sam Hartman is an author of the Kerberos specification and of a direction paper describing how to add support for systems like SAML to the Generic Security Services Application Programming Interface (GSS-API). Prior to founding Painless Security, Mr. Hartman served as the Chief Technologist for the MIT Kerberos Consortium and as a Security Area Director for the Internet Engineering Taskforce. Mr. Hartman has worked as product architect on products ranging from a consumer online banking application to enterprise infrastructure software and network attached storage.

Painless Security's evaluation is independent: prior to this engagement Painless Security had not reviewed or considered the Moonshot proposal. However Painless Security's independence is limited in that Painless Security is a potential partner for the implementation of Moonshot in future phases. The opinions expressed in this report are those of its authors and may not reflect the opinions of JANET(UK).

Moonshot is defined in two documents. The first describes a mechanism for carrying Extensible Authentication Protocol messages within the Generic Security Services Application Programming

Interface. The second describes how this mechanism can be used to simplify management of SAML metadata. In addition to these documents, the assistance of JANET(UK) staff and interviews with key stakeholders have been essential to the development of this evaluation.

This report is limited to considering whether a technical solution can be built that implements the proposal as described and on whether such a solution would work at a technical level to solve the two use cases. For the most part, policy, marketing and business concerns surrounding successfully deploying Moonshot are out of scope. In particular, Moonshot depends on managing trust relationships between a number of parties. Considering whether the appropriate business relationships can be put in place to provide information that can be trusted sufficiently to meet Moonshot's technical requirements is out of scope.

2.1. Goals

According to JANET(UK), the initial goal of this project is to build a test-bed that successfully demonstrates the application of the proposed architecture in the following scenarios:

1. Binding a federated identity to arbitrary application protocols;
2. Providing a scalable system for establishing trust between federated entities.

The success of the test-bed will be defined by these critical success factors:

- Technical feasibility: is the technology theoretically sound, and could existing systems be made compatible without requiring technically unreasonable modifications?
- Business feasibility: is the technology likely to yield benefits for the stakeholders, so that these stakeholders are able to develop business cases that justify the investment required to implement or use the technology?

The purpose of this report is to provide an independent analysis of the achievability of these critical success factors, with a focus on technical feasibility. It will be difficult to establish business feasibility until practical work on the test-bed and discussions with stakeholders have started. However, this report will highlight any unambiguous factors that may have a bearing on business feasibility.

2.2. Technical Feasibility

Painless Security believes that to succeed at a technical level, a test bed needs to associate a federated identity with an application session supporting the purposes of authentication, authorization, personalization and accounting. The following technical factors have been considered in evaluating the solution:

- Are the desired types of credentials supported?
- Are the desired types of applications supported?
- How does the solution address the metadata distribution use case?

2.3. Applicability

Moonshot makes several important assumptions about the environment of the client:

- Clients have EAP supplicant software installed and configured.
- Clients have configured their supplicant with some information about the sets of identities they use.

- Clients may be required to take some steps to enable support for Moonshot in their applications.

Thus, Moonshot is well suited to organizations that have the ability to configure the machines of all their affiliates. Moonshot may also be useful to experienced users who are able to follow directions and who see sufficient benefit in federated identity management to do so.

Moonshot is not well-suited to the needs of users who use machines that they do not control or that have not been configured to meet the needs of their organization. If the Moonshot technology is sufficiently successful, the necessary software may generally be available. However experience from the Kerberos community suggests that even though Kerberos software is generally available, even relatively simple configuration poses a significant obstacle to use of Kerberos on public machines. Moonshot is likely to see similar results.

This disadvantage is ameliorated somewhat because the applications that Moonshot targets work significantly less well from public machines or machines that have not been configured.

2.4. Credentials

A number of credential types are in common use in the European academic community. JANET(UK) has identified the following as common credential types.

- User name and password presented as a plaintext password
- User name and password presented using a cryptographic challenge/response protocol
- X.509 certificates
- One Time Password tokens such as Secure ID
- Kerberos

Ideally, a solution to the federation use case will support all these credential types. The current proposal can reasonably easily support all the credential types besides Kerberos. While there have been various proposals for using the Extensible Authentication Protocol with Kerberos, none have been standardized. There is no inherent technical problem developing the protocol and implementation necessary to use Moonshot with Kerberos.

2.5. Applications

Table 1 below describes applications identified by JANET(UK) and their support for the GSS-API. Both the support of the underlying protocol and of specific implementations is discussed.

Application	Protocol Support	Implementation Support
Jabber/XMPP	SASL	Good support in servers: Wildfire, Jabberd clients: Apple iChat, Pidgin No Windows client support
IMAP/SMTP (e-mail)	SASL	Outlook via SSPI, Apple Mail, Eudora, Evolution Good server support
SSH	RFC 4462 key exchange and user authentication	Open SSH, Putty, several commercial products
Remote Desktop (VNC, RDP)	None	RDP beyond Vista may support Kerberos
NFS	GSSRPC	Good Kerberos support on Linux, Solaris and commercial Windows products

Application	Protocol Support	Implementation Support
SMB/CIFS	Kerberos and NTLM only	Windows, Samba, Apple, others
SIP	None	None
HTTP	HTTP negotiate	Firefox, IE, Safari, Apache, many others

Table 1

3. Target Architecture

Moonshot proposes to combine two key security technologies: the Extensible Authentication Protocol (EAP) and the Generic Security Services Application Programming Interface (GSS-API) to create a solution for federated authentication. EAP provides federation and access to the mechanisms people use to authenticate while GSS-API simplifies integration with application protocols and applications. Moonshot is comprised of the following components:

- A GSS-API mechanism (or family of mechanisms) that encapsulates EAP authentication in GSS-API and provides security services after authentication completes
- A pass-through authenticator to take the GSS-API tokens, extract the EAP messages and use the RADIUS network to achieve federation
- Extensions to RADIUS to deliver authorization and personalization information back to the server
- Extensions to RADIUS and EAP methods to properly provide the level of mutual authentication required by GSS-API applications and establish trust

The basic approach has already been described in JANET(UK)'s drafts and slides. This feasibility report describes additional issues that need to be handled in the eventual design and implementation. Potential approaches to deal with these new issues are presented to demonstrate that the issues will require a refinement of the approach and do not present an intractable technical problem.

4. EAP in the Federated Model

EAP (RFC 3748) defines a framework for network access authentication. As typically deployed, there are three key entities in an EAP authentication:

- The peer: a client who is requesting access
- The authenticator: the resource from which access is requested
- The EAP server: an entity that interacts with the peer to perform an authentication protocol

Typically, the EAP server is integrated into a backend authentication server (or AAA server). EAP packets use some lower-layer protocol between the peer and authenticator. In typical deployments, the authenticator acts as a pass-through and RADIUS is used between the authenticator and EAP server.

The peer and EAP server are typically in the same administrative domain. The authenticator need not be.

EAP enables federated authentication because the peer and authenticator need not share any common credentials. The peer and EAP server typically directly share credentials. The authenticator and EAP server leverage hop-by-hop credentials in the RADIUS infrastructure. RADIUS also provides facilities to route messages and to map naming and authorization information.

RADIUS scales sufficiently to meet the needs of the European academic community. For example, RADIUS is used to handle authentication and authorization for roaming in wireless hotspots by major ISPs. RADIUS is also used for provisioning in the cable and DSL environments. Through eduroam the European academic community has specific experience with a RADIUS infrastructure. So, RADIUS and EAP make sense as a starting point for a federated access control solution.

JANET(UK) has specific resources that make this approach attractive. JANET(UK) has experience working with the Open1X EAP supplicant and FreeRadius, an open-source RADIUS server.

However there are also several challenges using EAP as the basis for a federated access control solution. As discussed in EAP Channel Binding, an EAP facility called channel binding is required in order to provide authentication to the specific service being accessed. The architecture for how to perform EAP channel binding is relatively immature. Several existing EAP methods do not support this architecture. It is likely that existing EAP supplicants and RADIUS servers do not include the necessary facilities. Thus, this approach depends on making EAP channel binding support available in authenticators, peers and backend authentication servers.

EAP naming is focused on network access. It will need to be extended to name the specific servers that are part of the federation. This will likely involve changes to peers, authenticators and backend authentication servers.

Several EAP methods have a fairly long authentication exchange involving multiple round trips. This is appropriate for initial network access where the authentication takes place at an initial network connection and need not be repeated. However, some of the intended applications may involve multiple authentications per second. For these situations, EAP methods with multiple round trips will perform unacceptably. Even if a particular method involves a small number of messages, the round trips with the backend authentication server will significantly increase latency. Modern network access situations such as hand off between access points or cells cause the same problems for the existing EAP use cases. As the situation improves in the broader EAP community, Moonshot should be able to take advantage of the improvements.

There is one question that will need to be thoroughly explored as part of phase 2. What changes

will need to be made to supplicants to support EAP use cases other than network access? While the answer probably doesn't directly impact technical feasibility, it will impact cost.

4.1. Summary

- EAP and RADIUS provide the scaling and robustness properties needed to form a basis for federated identity management.
- JANET(UK) and the European academic community have experience with this technology that makes it a natural choice.
- EAP has a relatively immature facility called EAP channel binding that will need to be expanded and implemented in order to meet GSS-API's mutual authentication requirements.
- Like other modern uses of EAP, Moonshot may require EAP fast re-authentication.
- Interactions with the EAP supplicant require additional study.

5. GSS-API and Application Authentication

GSS-API (RFC 2743) provides a framework that decouples application security operations from specific security mechanisms. The Simple Authentication and Security Layer (SASL) supports the use of GSS-API mechanisms in any protocol that uses SASL for authentication.

Between direct GSS-API support and SASL support, most application protocols include support for GSS-API. Thus, if a new security mechanism is made available through GSS-API, then it is potentially available in a large number of applications.

All of the desktop platforms including Windows, Mac, Linux and Java support GSS-API in some form. However, not all of these platforms support adding new mechanisms to their GSS-API implementation. In addition, Windows does support the over-the-wire protocol for GSS-API mechanisms, but it does not use the GSS-API's API natively. Instead, Windows uses the SSPI. Shims have been written so that applications that use the GSS-API can use SSPI. Also, third-party GSS-API libraries are available for Windows that do directly use the GSS-API.

Windows applications that are used in Active Directory environments typically have good support for the ways in which Active Directory uses GSS-API. However support for GSS-API in other environments is more limited.

There do not appear to be any candidates that are better than GSS-API for integration into applications. TLS APIs are not standardized so integration with TLS would require significantly more design effort to achieve standardized behaviour. In addition, using TLS for user authentication would be a fairly invasive change to some applications (although others support it today). Experience with Thunderbird and Pidgin suggests that GSS-API support can be added to an application as a relatively non-invasive change.

The decision to use GSS-API is sound. It will provide simplification of the design approach and will increase the probability that different vendors end up with compatible semantics. On server platforms, it may significantly reduce implementation costs as implementing for one application may accomplish much of the work of implementing for other applications. On the client, implementation cost savings of GSS-API will be much more modest than on the server.

5.1. Summary

- GSS-API provides integration with a wide range of application protocols.
- Implementation support for GSS-API is fairly wide-spread in the Active Directory environment and is more limited in other cases.
- GSS-API will provide significant savings in standardization effort and in implementation costs for server platforms but will provide only modest savings for client implementation.
- GSS-API can typically be added to an application implementation as a non-invasive change.
- Other choices would probably require more work to standardize and implement.

6. EAP and GSS-API Together

This section discusses design issues for an EAP GSS-API mechanism that affect the technical feasibility of that mechanism. With the exception of the multi-layer negotiation issue identified in the next section, solutions to all of these design issues are available: so far, none of these issues call into question the overall feasibility of the approach of using EAP for authentication and GSS-API as an application interface. However, a successful realization of this approach needs to budget time to address these design issues.

6.1. Negotiating the EAP Method

The draft specification currently describes a single GSS-API mechanism. The peer and authenticator exchange EAP messages. The GSS-API mechanism specifies no constraints about what EAP method types are used; text in the specification says that negotiation of which EAP method to use happens at the EAP layer.

However, EAP does not provide a facility for an EAP server to advertise what methods are available to a peer. At best, a peer can try each method it supports. Some lower layers do not permit a peer to use more than one method.

Even if EAP did provide a negotiation facility, providing multiple facilities to negotiate which security mechanism to use is undesirable. Section 7.3 of RFC 4462 describes the problem referencing the SSH key exchange negotiation and the SPNEGO GSS-API mechanism. If a client preferred an EAP method A, a non-EAP authentication mechanism B, and then an EAP method C, then the client would have to commit to using EAP before learning whether A is actually supported. Such a client might end up using C when B is available.

The standard solution to this problem is to perform all the negotiation at one layer. In this case, rather than defining a single GSS-API mechanism, a family of mechanisms should be defined. Each mechanism corresponds to an EAP method. The EAP method type should be part of the GSS-API OID.

That's problematic in an EAP and RADIUS environment. In order to start a RADIUS exchange, a NAS needs to send an Access-Request message. By that point, the NAS needs to know what EAP method to use. Even if RADIUS were extended to include a list of EAP methods in the reply, that would be too late for many protocols. For example, a SASL server needs to know what SASL mechanisms to offer before the client sends the initial GSS-API token. If the standard solution to the multi-layer negotiation were applied to EAP GSS-API, then the SASL server would need to know what EAP methods the EAP server supported before receiving any token from the client.

This issue will need to be addressed during the design and standardization of the mechanism. The current approach is one to consider if no better approach can be found. However, this approach means that the mechanism is open to bid-down attacks when used in conjunction with other security mechanisms. That will be seen as a weakness in the GSS-API community, although it may not be a significant weakness in JANET(UK)'s deployment.

6.2. Reliability

The EAP state machine is responsible for retransmitting EAP packets in order to maintain reliability. According to RFC 3748, when EAP is run over a reliable lower layer, then the retransmission time should be infinite. From this standpoint, GSS-API is a reliable lower layer: applications are responsible for managing context tokens and making sure they are reliably delivered.

However, the link between the authenticator and the backend authentication server is often unreliable. Typically it is RADIUS over UDP.

The bridge between GSS-API and RADIUS will be responsible for retransmitting EAP responses if an EAP REQUEST is not received. It is possible that if a GSS-API application takes too long in generating a response, the RADIUS server may drop state for the connection. In this case, the GSS-API bridge can generate an error. Specifying and correctly implementing this retransmission behaviour will require careful attention.

6.3. EAP Channel Binding

EAP authenticates a realm. The peer knows that it has exchanged authentication with an EAP server in a given realm. Today, the peer does not typically know which NAS it is talking to securely. That is often fine for network access. However privileges to delegate to a chat server seem very different than privileges for a file server or trading site. Also, an EAP peer knows the identity of the home realm, but perhaps not even the visited realm.

In contrast, GSS-API takes a name for both the initiator and acceptor as inputs to the authentication process. When mutual authentication is used, both parties are authenticated. The granularity of these names is somewhat mechanism dependent. In the case of the Kerberos mechanism, the acceptor name typically identifies both the protocol in use (such as IMAP) and the specific instance of the service being connected to. The acceptor name almost always identifies the administrative domain providing service.

An EAP GSS-API mechanism can use the EAP channel binding facility to match this level of entity authentication. EAP channel binding is currently defined by draft-ietf-emu-chbind. It provides a framework under which an EAP peer securely communicates information to the EAP server through the EAP exchange or a secure association protocol. The authenticator passes attributes to the backend authentication server. The backend authentication server performs a consistency check to make sure information received from the peer is consistent with information received from the authenticator.

The channel binding document points out that in many cases, a local database will be required at the authentication server to constrain what information can be provided by the authenticator. Most EAP GSS-API deployments will require this database. In particular, the backend authentication server will need to verify that:

- The authenticator correctly indicates its realm to the peer
- The authenticator's realm is intended to provide the type of service being provided (E.G. if an example.com authenticator claims to be providing imap service, example.com is expected to provide imap service to local users)

In addition, some component in the realm of the authenticator needs to verify that the authenticator is correctly asserting its identity. Meeting GSS-API's requirements for authenticating the endpoints places several requirements on the system:

- Either the GSS-API binding or the EAP methods must support channel binding exchange.
- The GSS-API peer must include the name of the acceptor in its channel binding data.
- The GSS-API mechanism must include the name of the acceptor in data passed to the backend authentication server.
- A mechanism in the realm of the authenticator must verify the name supplied
- Support in the backend authentication server of the home realm for constraining acceptable acceptor names

Determining the realm of the acceptor places additional requirements discussed in the next section. Unfortunately these requirements involve changes to the RADIUS servers involved in the authentication.

At a technical level, meeting these requirements does require additional design, standardization and implementation, but appears to be reasonably feasible to accomplish. However there is also a policy requirement that the home realm needs to be able to trust the visited realm to verify the server's identity. Exploring the implications of this policy requirement is out of scope for this technical feasibility analysis.

6.4. Naming and Authorization

GSS-API has a number of APIs designed to manipulate names and name attributes for authorization. Even so, applications often end up manipulating GSS-API names in at least somewhat of a mechanism-specific manner. We have proof from SAP R3 that a GSS-API application can be written with no knowledge of mechanism-specific naming. Many applications take the simpler approach of building mechanism-specific naming assumptions. For example it is common for applications to expect names in the form of Kerberos principals to be used on access control lists. The applications will need to change in order to take advantage of the EAP GSS-API mechanism.

Even when an application avoids mechanism-specific details, we will need to invent a layer to express naming and authorization sufficient to meet application needs.

Users can be named using the GSS_NT_USER name type. Names of this form typically take the form *user@REALM* which is approximately the same as the network access identifier (NAI) used in by the IETF AAA technologies.

Kerberos services are typically named using the GSS_NT_SRV_HST form which takes the form SERVICE@HOST. Supporting this name type in calls to gss_import_name is important because it is required by many SASL profiles. This name form leaves open the question of which authentication realm should represent a given host. For example, both the good.example.com and evil.example.com realms could claim to represent imap@mail23.example.com. In a federated environment, confirming that the client is using a service from the appropriate authentication realm is critical to mutual authentication.

The realm can be exchanged as part of channel binding. However the backend authentication server needs to know what realm is appropriate for a given service name. Identifying how this is handled will be an important design consideration.

The GSS-API mechanism should also support a name type that includes both the realm and host, similar to the GSS_NT_KRB5_PRINCIPAL name type that RFC 1964 supports. The advantage of this is that when a client has high confidence in the realm that should be expected, the client can communicate this to the backend authentication server.

6.5. Security Layers, GSS-API Channel Binding and PRF

EAP provides two keys: the master session key (MSK) and extended master session key (EMSK) that can be used by the GSS-API mechanism.

GSS-API provides several services that use keys. The simplest is the gss_wrap API, which allows an application to integrity and confidentiality protect a message. The current draft proposes using RFC 4121 tokens for this service. That approach works well and has been used by other proposed GSS-API mechanisms. In addition, using RFC 4121 makes it clear how to support gss_wrap_iov, an extension to GSS-API needed by DCE RPC and other protocols used in the Active Directory

environment. Since RFC 4121 framing is implemented in the Solaris, Windows and Linux kernels, reusing this framing will make it easier for kernel-mode GSS-API applications such as CIFS and NFS to take advantage of this technology. For Kerberos at least, gss_wrap is used by the SASL profile. However, it seems like a new GSS-API mechanism would use the GS2 SASL profile, which relies on TLS for confidentiality and integrity.

GSS-API provides a facility called GSS channel binding. Channel binding data is passed into gss_init_sec_context and gss_accept_sec_context. The initiator and acceptor verify that the channel binding data provided on both sides of the connection is the same. Currently the draft does not describe how to handle this facility. Two approaches are possible. One is to leverage EAP channel binding and include the channel binding data in data sent to the backend authentication server for verification. Another approach is to use an EAP-derived key to integrity-protect an exchange of channel binding data. The second approach may be more robust because it does not depend on the backend authentication server actually verifying consistency of channel binding data. This facility is critical to using GSS-API with the new GS2 SASL mechanism, so supporting GSS-API channel binding is critical to several applications in the federated authentication use case.

GSS-API also provides a pseudo-random function as described in RFC 4401. This can be used to set up a key hierarchy. Since RFC 4121 is used, using the implementation of this function defined in RFC 4402 seems reasonable. This facility is relatively new. Future versions of the AFS distributed file system are expected to use this extension for establishing a key hierarchy. In addition, integration with SAML should use this mechanism.

If GSS-API is going to be used with certain Microsoft protocols, including CIFS and the new extended negotiation protocol, then the GSS-API mechanism will need to expose a context key for these Microsoft protocols to use.

As part of the design, the mechanism specification needs to clearly decide on which EAP key is to be used. Using the MSK may be fine. Using the EMSK may provide a more clearly defined key hierarchy which has cryptographic advantages. However, the EMSK cannot be exported from the backend authentication server. Instead, the backend authentication server would need to derive keys from the EMSK and provide them to the authenticator.

6.6. Summary

- Moonshot specifications will need to consider the multi-layer negotiation problem and decide on a mechanism for selecting an EAP method
- Specifications should carefully describe retransmission behaviour of the GSS-API to RADIUS bridge.
- Moonshot will need to support the GSS_NT_USER name type and the GSS_NT_SRV_HST name type.
- Moonshot will need a mechanism for confirming that a particular visited realm is appropriate for a given host.
- Supporting GSS-API's level of mutual authentication places several requirements on the system:
- Either the GSS-API binding or the EAP methods must support channel binding exchange.
- The GSS-API peer must include the name of the acceptor in its channel binding data.
- The GSS-API mechanism must include the name of the acceptor in data passed to the backend authentication server.

- A mechanism in the realm of the authenticator must verify the name supplied
- Support in the backend authentication server of the home realm for constraining acceptable acceptor names

7. Authorization and Personalization

Moonshot proposes to transport a SAML assertion from the RADIUS server to the GSS acceptor. This assertion could be made available to applications using the GSS naming extensions. In order to do this, the RADIUS server will need to determine what SAML attributes to release to a given acceptor. Members of the UK Access Federation already have identity providers that implement attribute release policies. The RADIUS server should rely on these existing identity providers.

No existing applications will support this mechanism for conveying a SAML assertion. Adding support for retrieving the SAML assertion is relatively easy. However depending on what personalization and authorization decisions are desired, extending the application to support this may be complex. This is an inherent complexity in extending an application to support attributes in a federated manner; Moonshot will be no easier or harder than any other approach.

In addition, GSS-API applications expect to learn the username of the subject. Attribute release policies may not permit releasing the subject's username. Usernames may be used in a number of ways. For example, users expect to be able to put a username on an access control list and for that user to gain access. GSS-API is sufficiently flexible that it can work in this situation. When the username and realm can be released, the experience should be as usable as any other GSS-API mechanism. However, minimizing the decrease in usability when the username cannot be released will be a design challenge.

8. Metadata Maintenance

The metadata maintenance proposal has undergone significant revision as this report was prepared. At this point it appears that there are two primary ways Moonshot can help with metadata.

The first is to leverage the RADIUS infrastructure to gain trust in metadata. A GSS-API name form can be created that cryptographically binds to a specific instance of metadata. The metadata consumer authenticates to the target entity using a GSS-API name referring to the target entity's metadata instance. RADIUS servers close to the target entity and to the consumer's identity provider both have an opportunity to validate that the metadata instance should be trusted. In addition, the target entity can confirm by its choice of acceptor name that the metadata is something it trusts. This means that no credential infrastructure beyond the RADIUS and EAP credentials is required to build trust in a SAML federation. In addition, online revocation of metadata is supported by this system.

Moonshot may also simplify the key management problem associated with SAML. The EAP GSS-API mechanism can be used to establish a pair-wise key between the metadata consumer and target entity. This key can be used to integrity-protect or confidentiality-protect SAML exchanges, potentially removing the need to include asymmetric keys in the metadata.

These facilities can be applied selectively. For example, metadata can be signed and validated via EAP-GSS. Consumers for whom management costs would be lower managing a PKI can do that while other consumers can use EAP-GSS. Obviously, a single federation can achieve the most cost savings by minimizing the number of credential infrastructures that are maintained. However, the optional nature of the facility should enable deployment in situations where it makes operational sense.

9. Work Plan

Two projects will run in parallel to develop the test bed. The standardization effort will conduct necessary work to specify protocols used by the system in sufficient detail that products from multiple vendors can work together. This work is expected to be split across the IETF and OASIS. In addition, the test bed itself will be developed.

9.1. Standardization Work Required

One key standardization activity will be the specification of an EAP GSS-API mechanism starting from the existing proposal. This work includes:

- Expand the EAP applicability statement beyond network access.
- Specify the context establishment messages.
- Specify the key hierarchy and how it is used to generate RFC 4121 tokens for per-message services.
- Specify how GSS-API channel binding is implemented.
- Describe what GSS-API name types are supported and how they are represented in the EAP/AAA system.
- Specify the behaviour of a pass-through authenticator that bridges a GSS-API application to the AAA infrastructure.
- Describe requirements for EAP channel binding.
- Describe how multiple EAP methods will be supported.
- A RADIUS attribute for carrying a SAML authorization assertion from the backend authentication server to the authenticator needs to be specified. In addition, how the GSS-API naming extensions are used to access this data needs to be specified. A mechanism for mapping GSS names used by the EAP GSS mechanisms to SAML entity identifiers needs to be specified.
- A profile of SAML for this authorization assertion needs to be selected or developed. In order to do this, more clear understanding of what attributes or claims about identity are useful to service providers needs to be understood.

9.2. Implementing a Proof-of-Concept Demonstration

To be successful the proof-of-concept needs to demonstrate working with HTTP and with the metadata maintenance problem. It would be strongly desirable to demonstrate utility with another application such as SSH.

This section enumerates high-level milestones for each part of the project. These milestones were developed in fairly close cooperation with JANET(UK) staff, although the results reflect the opinion of the authors of the report and may not reflect those of JANET(UK).

9.2.1. Client Infrastructure

- Choose a GSS-API library (low effort)
- Write GSS-API context establishment to call into an EAP supplicant (high effort)
- Initial context calls into supplicants

- Accept context calls into RADIUS library
- Design representation of mechanism names
- Acceptor name sent by client via supplicant to EAP server as part of EAP channel binding
- Acceptor name sent to RADIUS as part of accepting a context
- Name attributes and initiator name received from RADIUS
- Design mechanism-specific credential representation and semantics for acquiring credentials
- Implement context key and sequencing establishment
- Add supplicant support for a GSS lower layer (high effort)
- Implement EAP channel binding support in the supplicant (medium effort)
- Implement channel binding in an EAP method
- Extend the supplicant UI to describe what application and service requests authentication (medium effort)

While preparing a work plan, one goal was to provide support for Moonshot to JAVA platform applications. No current use cases at the proof-of-concept phase require this work. The milestones are enumerated below as this support may be desirable for future use cases.

- Support from within JAVA by extending JAVA native GSS support
- Write naming extensions for Java GSS SSPI (high effort)
- Bridge naming extensions through to native Java GSS mechanisms (medium effort)
- Write PRF for GSS SSPI (medium effort)
- Bridge PRF through to native mechanisms (low effort)
- Provide way to distribute modified security mechanisms for test bed users (high effort)
- Investigate support for EAP GSS name and credential types

9.2.2. Application Integration

- HTTP
    - Expand negotiate to deal with multi-round-trip mechanisms (browser) (high effort)
    - Expand negotiate to deal with multi-round-trip mechanisms (mod-auth-kerb) (high effort)
    - Expand mod-auth-kerb to deal with non-Kerberos (low effort)
    - Expand mod-auth-kerb to expose the GSS context including name attributes and the PRF to other parts of Apache (medium effort)
    - Expand Shibboleth service provider to take external authentication and authorization from earlier in the stack (medium effort)
    - Document constraints regarding proxies, load balancers and the like (low effort)
- Metadata Maintenance
    - Write SAML key exchange web service (medium effort)
    - Write SAML key exchange and metadata trust client (medium effort)

  - Implement metadata name support in GSS-API (medium effort)
  - Implement AAA server support to revoke metadata and confirm metadata names (low effort)
- SSH
  - Add support for our GSS mechanism(s) to OpenSSH GSS patches (low effort)

9.2.3. Authentication Server Work

- Add support to EAP module and EAP methods to expose channel binding data as attributes (medium effort)
- Compare channel binding data received from peer against channel binding data received from authenticator (medium effort)
- Constrain acceptable channel binding data based on policy (high effort)
- Proxy near authenticator confirms acceptor name is correct
- EAP server or proxy near EAP server confirms that realm of acceptor is acceptable
- EAP server or proxy near EAP server confirms that realm of acceptor matches host of acceptor for GSS_NT_SRV_HST
- EAP server validates metadata name form
- Support requesting attributes from an asserting party and including in the RADIUS response (low effort)

10. Alternatives

Several comparison criteria emerged when looking at proposed solutions to federation of non-web applications.

The first criterion is whether the client needs to talk directly to the authentication infrastructure or whether the conversation with the authentication infrastructure is tunnelled. Kerberos deployers have experienced significant difficulty using Kerberos as part of a federation strategy because organizations are reluctant to directly expose Kerberos KDCs to the Internet and because even when they do, firewall policies near the client may pose problems. So, from the standpoint of these deployments, a mechanism like EAP that provided tunnelled authentication would be desirable. In contrast, the SAML IdP community has exposed their identity providers to the web so that can be used as part of SAML Web SSO. In these deployments, tunnelled authentication would not be viewed as a significant advantage.

Another criterion for comparison is how the solution provides protection against phishing and how it provides the mutual authentication service. With the web we've seen that as value can be gained from attacking a credentials infrastructure, criminal elements will attempt to gain access to credentials. If properly managed, the security technologies used on the web such as TLS and PKI can defend against phishing attacks at a technical level. However, in practice, these technical defences do not work: users are too willing to accept invalid certificates or to ignore other indicators. s In contrast, systems like Kerberos that exploit pre-existing relationships to provide their mutual authentication and phishing defence may reduce the effectiveness of these attacks.

Another area of difference is whether a web browser is used for initial authentication. Many websites have indicated a strong requirement to control the user interface of the login experience. For example in the SAML Web SSO approach, the relying party presents a web interface, chooses when or if an identity provider is selected, and manages the selection of that identity provider. The identity provider manages the interface used to log the user in. In contrast, with a system like EAP or CardSpace, a trusted component of the local environment manages the authentication process. Using a local component provides less opportunity for remote parties to brand the interaction. This may cause the user experience to be perceived as less cohesive. Support costs may be increased because even though the user believes they are logging into the relying party, the relying party does not control the error messages and may have difficulty documenting all the interactions. This can be especially true if there are multiple different local components a user may have with different user interfaces. However, there are significant advantages to using a local component for authentication. Doing so makes it significantly easier to improve the cryptographic security of the credentials used. A challenge/response protocol such as EAP GPSK or Kerberos can be used instead of plaintext password protocols. Smart cards and cryptographic protocols locally secured by biometric authentication are easier to offer. Also, many security practices try to use a consistent user interface for local authentication to distinguish it from an attacker. For example, Windows uses a secure attention key to unlock the screen saver and the Identity Selector uses a secure desktop. However, usability research suggests that these mechanisms are not particularly effective: users do not tend to determine whether they are interacting with a trusted user interface before providing sensitive information such as passwords. A system that uses web browsers for initial authentication is completely unsuited to a number of deployments. For example, when one service is authenticated to another, a web browser does not make sense. The metadata maintenance use case is an example of a deployment where web browser authentication would be unsuitable.

Another significant criterion for comparison is how much new development is required and how well the mechanism will fit into existing applications.

10.1. Information Card

Microsoft's Information Card provides a mechanism for carrying identity claims based on the WS-* stack. This provides a complete solution for the web. In order to use this solution with applications other than the web, some mechanism for using WS-* security tokens in applications would be needed. One approach would be an Information Card GSS-API mechanism. Security Token Servers (STS) would be needed in order to issue security tokens based on initial authentication.

Information Card has been implemented both with local authentication and using a remote website to manage the authentication. The Windows implementation uses a local identity selector. Information Card requires direct access by the client to the authentication infrastructure. Painless Security has been unable to determine to what extent Information Card supports mutual authentication.

Standardizing and implementing an Information Card GSS-API mechanism would be reasonably involved. It would share some of the work of Moonshot: a solution for per-message security services, channel binding, and the PRF would be required. In addition, applications would need a WS-* stack profiled to work in an interoperable manner. Aligning Information Card's identity claim requirements with GSS-API applications' naming requirements might prove difficult.

Information Card's major advantage is its advanced claims-based notion of identity and the potential of Microsoft's support.

10.2. Raw SAML GSS-API Mechanism

Over the years, there have been multiple proposals to carry a SAML assertion in a GSS-API or SASL exchange for authentication. The simplest of these approaches direct a user to a web browser to initially authenticate to an IDP and send an obtained assertion for authentication. The most complicated involve a new GSS-API mechanism with an unspecified web service for obtaining SAML assertions.

The simpler variants of this approach are particularly likely to appeal to organizations trying to reuse existing Web SSO infrastructure for other applications. These simpler variants do not provide many of the desirable properties of Moonshot such as per-message security and channel binding. In many of these mechanisms the possession of a SAML message is all that provides authentication. In effect, the SAML message acts as a password or one-time password (depending on whether replay prevention is used). Moonshot provides stronger security than this use of SAML.

Other proposals, including one investigated in the IETF by Sun, provide a significantly more advanced use of SAML. These proposals would provide more of the Moonshot services, although they required implementation complexity and deployment complexity similar to or in excess of Moonshot. The proposals in this space that Painless Security is aware of are at earlier phases of development and would require as much or more standardization work than Moonshot.

10.3. Backend GSS into an Authentication Server

One concern about Moonshot is that it involves many security technologies. One way to reduce the number of security frameworks involved would be to create a way of splitting the role of a GSS-API acceptor so that the GSS-API mechanism could run on an authentication server and the acceptor would not need to have code specific to the client's credential type.

Transporting GSS-API tokens over RADIUS would be very similar to sending EAP messages through RADIUS. Doing so would require additional standardization work because GSS-API RADIUS transport is not standardized today. The major complexity of this approach is dealing with per-message security and the PRF. For any given GSS-API mechanism, performing a context

transfer from the authentication server to the acceptor is a small matter of standardization and implementation. Similarly supporting all mechanisms that ultimately use RFC 4121 tokens would not be too involved from a standardization effort, although it would pose significant implementation complexity as a GSS-API library would need a common facility to determine if a mechanism used RFC 4121 tokens and to extract the context in a standard form. Similarly, differences in the set of encryption types supported by the acceptor and authentication server might pose deployment concerns.

Stackable pseudo-mechanisms provide a generic approach for solving the per-message security and PRF problems. Two mechanisms are composed together: an outer per-message mechanism and an inner initial context establishment mechanism form a single mechanism. When context tokens are exchanged, the outer mechanism simply passes the token to the inner mechanism. After the context is established, the outer mechanism uses the inner mechanism's PRF to establish keys for the outer mechanism. The abstraction between the outer and inner mechanisms falls at the boundary between the acceptor and authentication server. Several issues would need to be examined to make this approach work. One is making sure that security provided by the outer and inner mechanisms is sufficiently similar to meet government requirements. Another concern is how to handle mechanisms such as Simple and Protected Negotiation (SPNEGO) that are already stacked mechanisms. Would the security token mechanism be stacked outside SPNEGO or inside SPNEGO? Outside poses significant problems but inside makes it challenging to confirm that the eventual negotiated mechanism will support the acceptor/authentication-server split. The stackable pseudo-mechanism approach does require changes to all clients: whether a pass-through acceptor is used or whether the acceptor is combined is exposed to the client in the offered mechanism OIDs. Implementing the stackable pseudo-mechanism approach is likely to be relatively tedious with current GSS-API frameworks.

Even if the stackable pseudo-mechanism solution or another solution to the per-message security and PRF problem is adopted, this approach does not solve Moonshot's objectives today. GSS-API does not have a good set of interoperable implementations for the username/password case where the server learns the password. Other than Kerberos, interoperable solutions for X.509 credentials and one-time-password credentials are limited. Two solutions are at various levels of standardization: SPKM and PKU2U are GSS-API mechanisms that support X.509 credentials. Painless Security is not aware of a PKU2U implementation. There are several SPKM implementations, but when the IETF considered revising SPKM, the consensus of the parties involved was to abandon it. One concern that lead to this decision was a lack of interoperable implementations.

This approach does present a somewhat simpler and more elegant architecture. However it does not seem like it is a viable alternative to the Moonshot core objectives today. The approach of using stackable mechanisms and GSS proxying may be a useful tool in the future. For example, it could be used to provide rapid re-authentication. As Moonshot goes forward it is desirable to allow for the possibility of deployments where GSS rather than EAP is proxied. Work done to standardize or implement GSS proxy should be associated with a specific and sufficient use case. Work to make sure the design is sufficiently general seems justified by the architectural principle of extensibility and by potential benefits such as re-authentication.

10.4. Backend EAP into Kerberos pre-authentication and use IAKERB

During discussions surrounding Moonshot's technical feasibility, several parties raised the concern that applications are more likely to support GSS-API Kerberos than generic GSS-API. If Moonshot could be made more like Kerberos then integrations with clients and client applications might be easier. Ideally, applications would not even realize they were using Moonshot, but instead would believe they were using Kerberos.

Achieving this while maintaining Moonshot's critical objectives means using EAP for initial authentication and Kerberos for application-level authentication. There are two challenges with this approach. First, Kerberos does not support EAP today. Second, federation with Kerberos is problematic in part because it requires exposing Kerberos KDCs to the firewall.

IAKerb (draft-ietf-krb-wg-iakerb) is an ongoing work item of the Kerberos working group in the IETF to solve the second problem. The Kerberos initial authentication is proxied as part of a GSS-API exchange. The first problem could be solved by adding EAP authentication to the Kerberos pre-authentication exchange: a subject could exchange EAP messages with a Kerberos KDC and, if successful, EAP authentication would result in issuance of a Kerberos ticket-granting-ticket. For Moonshot-like deployments, the KDC should live close to the relying party, not the authentication server, enabling the use of RADIUS rather than Kerberos cross-realm for federation. Then a service ticket can be obtained and a normal Kerberos GSS-API authentication performed.

There are implementations of IAKerb. Kerberos implementations other than Microsoft typically expose an interface for pre-authentication plugins so that EAP support could be added dynamically to clients and close to dynamically to KDCs. Luke Howard has demonstrated a facility for including SAML assertions in a Kerberos ticket; this facility could be used to include the SAML message received from RADIUS into the issued ticket, where it could be accessed by the application. While Windows does not support plugins for pre-authentication, Windows Vista and beyond support a mechanism where an external Kerberos implementation can obtain a ticket-granting ticket and provide that ticket to Windows.

Unfortunately, by using IAKerb rather than the existing Kerberos mechanism, some of the same incompatibilities that would face Moonshot would be introduced. Also, such a deployment would require one Kerberos identity per realm containing a utilized service. Some Kerberos implementations would function well in this mode; others work poorly with multiple Kerberos identities. In addition, Windows does not support IAKerb today. It seems likely that it will in the future.

In summary, it seems possible to design something that is very similar to Moonshot that is much closer to Kerberos. There would be increased complexity. Application integration would be simpler, although not as simple as for Kerberos itself. The main downsides of this approach would be the requirement to deploy KDCs, increased complexity, and an increased number of round trips for initial authentication to a service. This approach would provide better capability for providing reduced-user-experience mechanisms to federate applications when the client application does not directly support Moonshot. Also, this approach would solve the fast re-authentication problem.

From an application and user standpoint, this approach is very similar to Moonshot as specified today. The decision of whether to take some components or ideas from a Kerberos integration into the Moonshot proposal can probably be deferred until summer 2010. Because the application integration problems are a serious concern, the Kerberos integration should be studied at least enough to understand whether aspects can be borrowed.

11. Conclusions

This report analyzes the feasibility of the Project Moonshot Technical approach. Moonshot proposes to combine several existing security technologies to bring federated identity management to applications beyond today's web-browser use cases. The Extensible Authentication Protocol (EAP) and RADIUS infrastructure is used to provide federation. The Generic Security Services Application Protocol Interface (GSS-API) is used to integrate Moonshot into existing and future applications. The Security Assertion Mark-up Language (SAML) is used to provide attribute exchange and personalization. Moonshot technology may also simplify the metadata maintenance problem faced by SAML federations such as the UK Access Management Federation by re-using an EAP credential infrastructure for establishing trust in metadata and by reducing the need for key management associated with metadata.

In preparing this evaluation, Painless Security extensively analyzed the interactions between EAP, GSS-API and SAML. In addition, interactions between the actors in the system including clients/subjects, relying parties/servers, the RADIUS servers, and identity providers were carefully considered. Based on this analysis, the approach appears technically feasible. In addition, the resulting technology should substantially address both the use cases of federation beyond web applications and metadata maintenance. JANET(UK) identified a number of applications for which Moonshot should enable federation. Moonshot is a good fit for most of these applications. However today's voice over IP protocols and remote desktop protocols do not support GSS-API for security services. Moonshot, as conceived today, cannot federate these applications. In addition, while Moonshot can support most of the types of credentials identified by JANET(UK), Kerberos would not be supported by EAP as it exists today. Neither the applications nor the credentials that cannot currently be supported were identified as critical.

GSS-API, EAP and SAML are all mature technologies. They meet the robustness and scalability requirements for a production federation. Based on the analysis conducted, these technologies were the best fit considered for federation of applications for systems under the control of an organization that can provide software and configuration. These technologies do not provide a good user experience for public machines or machines that have not received some configuration. However, the applications targeted for Federation by Moonshot typically also require some degree of configuration.

While the underlying technologies are mature, Moonshot extends all the technologies in order to provide optimal integration: standardization regarding EAP, RADIUS, GSS-API and SAML will be required. The vast majority of this standardization activity involves extensions such as definition of new GSS-API mechanisms and new SAML bindings intended to happen as the technologies are adopted into new environments. Some of the standardization activities such as the expansion of EAP's applicability statement are closer to the core of the technologies. However, even these activities are outgrowths of ongoing discussions in the appropriate standards bodies. While there are open issues, none of these issues appear to be bigger than those tackled routinely as part of standardization efforts of this scope. Political negotiations will doubtless be required. At least with regard to IETF activities, none of the standardization activities appear to depend on courses of action that have been decided against or ruled out.

Moonshot's implementation will gain significantly from the use of existing security technologies. Nevertheless, it will be difficult. Using these three technologies together is novel and involves integrating new systems. Even though all three technologies are generic at a conceptual level, specific uses of the technologies tend to make assumptions that Moonshot challenges. For example, applications may make assumptions about what mechanisms are used or about how identity is represented. So, while reuse of technology will produce significant savings, Moonshot will not achieve the ideal result where Moonshot could be integrated into a system and applications or

environments unaware of Moonshot would work with it without change. In particular, in most cases, clients, servers, RADIUS servers and EAP implementations will all need to change in order to take advantage of Moonshot. The design will minimize these changes.

During the technical analysis, one critical non-technical factor to Moonshot's success emerged. In order for Moonshot to be successful, it will need to be available to client applications. This means support by vendors of platforms such as Windows, Mac OS X, and Linux as well as vendors of popular application frameworks and scripting environments. One approach was suggested for making client application integration easier: make Moonshot more similar to Kerberos. In theory, the more Kerberos technology that is reused, the more existing integration of Kerberos into GSS-API applications may benefit Moonshot. This approach is explored in the discussion of alternatives. No specific changes are recommended, although exploring ways to improve simplicity of application integration will be important. Even if Moonshot is changed to be more like Kerberos, the essential characteristics of the proposal will not change. Making changes such as those discussed in the Kerberos integration alternative is unlikely to significantly impact cost unless the change is made late in the process or multiple paths are explored to support multiple different deployment situations.

Further analysis is recommended as Moonshot progresses. The interactions between Moonshot and the EAP supplicant need more study. Examining methods to reduce application integration cost including ways to leverage existing Kerberos integration will continue to be important. Understanding how Moonshot and Oauth can compliment each other or where their utility overlaps will likely be important to positioning and optimizing Moonshot. This additional analysis is not expected to call into question Moonshot's feasibility but will improve the design and aid in positioning Moonshot.