Brigham Young University
BYU ScholarsArchive

All Theses and Dissertations

2008-07-14

Simple, Secure, Selective Delegation in Online Identify Systems

Bryant Gordon Cutler
Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Computer Sciences Commons

BYU ScholarsArchive Citation

Cutler, Bryant Gordon, "Simple, Secure, Selective Delegation in Online Identify Systems" (2008). *All Theses and Dissertations*. 1474.
https://scholarsarchive.byu.edu/etd/1474

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

SIMPLE, SECURE, SELECTIVE DELEGATION IN ONLINE IDENTITY SYSTEMS

by

Bryant Cutler

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

August 2008

Copyright © 2008 Bryant Cutler

All Rights Reserved

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Bryant Cutler

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Phillip J. Windley, Chair
Date	Kent E. Seamons
Date	Mark Clement

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Bryant Cutler in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Phillip J. Windley Chair, Graduate Committee

Accepted for the Department

Date	Kent E. Seamons Graduate Coordinator

Accepted for the College

Date	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

SIMPLE, SECURE, SELECTIVE DELEGATION IN ONLINE IDENTITY SYSTEMS

Bryant Cutler

Department of Computer Science

Master of Science

The ability to delegate privileges to others is so important to users of online identity systems that users create ad hoc delegation systems by sharing authentication credentials if no other easy delegation mechanism is available. With the rise of internet-scale relationship-based single sign-on protocols like OpenID, the security risks of password sharing are unacceptable. We therefore propose SimpleAuth, a simple modification to relationship-based authentication protocols that gives users a secure way to selectively delegate subsets of their privileges, making identity systems more flexible and increasing user security. We also present a proof-of-concept implementation of the SimpleAuth pattern using the sSRP authentication protocol to demonstrate the generality of our technique.

Contents

Chapter 1

Introduction

As use of the world-wide web becomes more tightly integrated into the daily lives of people around the world, online identity systems become increasingly important. Online identity to date has been ad hoc, site-specific, and generally insecure. Recently, new online identity systems and authentication protocols including OpenID [9] and Identity Metasystem [8] have offered more workable solutions that are user-centric rather than site-specific. Unfortunately, these identity systems have some severe limitations relative to real-world identity; one limitation is that they do not include explicit support for delegation.

The primary benefit of real-world delegation is increased flexibility. Delegation allows users to accomplish tasks in a more efficient or convenient way than would otherwise be possible. For example, a manager might delegate responsibility for an equipment purchase to an assistant, allowing him to use a company's account. In this way, the manager ensures the completion of a task without doing the work himself, and without the additional cost of maintaining a separate purchasing account for each employee.

The increased convenience and efficiency of delegation are not just desirable qualities: they are vital to the success of identity systems. In practice, users will willingly share authentication credentials - even at the risk of security breaches - to delegate to coworkers, friends and family members. This security problem is exacerbated by single sign-on systems, in which a single credential provides access to many resources. Delegation will occur whether officially sanctioned or not; thus, explicitly adding secure delegation capabilities to an identity system will increase users' security by allowing them to comply with the "principle of least privilege" [10]. Restriction of delegate privileges and elimination of password sharing reduces the risk of delegating by limiting the damage a malicious or incompetent delegate can inflict.

This privilege control is especially important because the web offers users new delegation scenarios where the normal rules of human conduct may not apply. For example, when a professor delegates access to his email account to a secretary by sharing his password, there is a strong social disincentive for the secretary to misuse the professor's privileges. Web users often share passwords with friends and family members with no ill effects. However, if a user wishes to delegate access to the same private data to a computer system or service rather than a person, there is no equivalent basis for trust.

Adoption of authentication protocols that include delegation would make it easier for users to delegate to other users and services, without compromising their credentials or giving untrustworthy delegates too many privileges. Example use cases include:

- *Delegation to other users* In the simplest case, a delegation-enabled authentication protocol allows users to share their privileges and private data with other users; the primary advantage over current ad hoc delegation mechanisms is the cessation of password sharing. For example, a student might delegate access to his online medical records to his parents, without sharing the single sign-on credential he uses to access a social network.
- *Multi-step delegation* Users may give delegates not only user privileges, but the ability to further delegate a subset of those user privileges to others. This enables the use of a dynamic "trickle-down" user-mediated access control model in situations where more static, role-based access control is inappropriate. For

example, the deans of the colleges within a university might each delegate different privileges to their assistants, reflecting differing managerial styles; those assistant deans may in turn grant different privileges to each department chair they supervise.

- *Self-delegation* Users can protect themselves from insecure computing environments by using pseudonymous personas to whom they have temporarily delegated limited privileges. This decreases user risk: even if the pseudonym is compromised, its time or usage limits and restricted privileges will reduce the damage a successful attacker can cause. For example, a business traveler might arrange a one-time-use pseudonym with read-only access to his webmail, so that he can safely check his email from a cybercafe.
- *Delegation to a service* Users can delegate privileges to online services as well as other users, making use of "mashup" services more secure. For example, a user might give a mashup access to his webmail contact list without giving it the ability to send email from his address.

With the goal of safely making online services more convenient and efficient, we propose a new pattern called SimpleAuth. SimpleAuth is a simple modification to relationship-based authentication protocols that allows users to selectively delegate their privileges, making identity systems more flexible and increasing user security.

Chapter 2

Delegation Concepts

2.1 Taxonomy and Terminology

Online, identity begins with a *relying party*, or *RP*, which represents the entity (for example, a web application) that provides a service to users. A *user* is a person with access to the RP's functionality. Users are represented by digital identifiers (a single user may have several identifiers for use in different contexts) which are managed through an *identity system*, and verified by an *authentication protocol*. Recent identity systems and their included authentication protocols strive to be *user-centric*; they give users (rather than administrators) as much control as possible over the use of user identifiers and personal data.

Bhargav-Spantzel *et al.* proposed a taxonomy for user-centric systems with a division between credential- and relationship-based identity systems [4]. *Credential-based systems* rely on long-lived, hard-to-forge, securely-stored credentials like keys and certificates, typically managed by the user with a smart client. These systems theoretically work offline; however, their security against compromised credentials depends on periodic consultation of online, up-to-date revocation lists.

In contrast, authentication in *relationship-based systems* is based on proof of a relationship between a user and an online verifying entity called an *identity provider*, or *IdP*. IdPs must be online and available to participate in every authentication transaction. This introduces a single point of failure to the system, but it makes revocation trivial and allows users access to their identities from any web-connected device.

While some protocols like Identity Metasystem [8] are hybrids, with both long-lived credentials and online identity providers, most identity systems and authentication protocols fall firmly into just one of these categories. Traditional password authentication and systems based on the public key infrastructure (PKI) are credential-based. Relationship-based systems, which are the basis for SimpleAuth, include OpenID [9] (used in our earlier work [2]) and the sSRP protocol [5] (described in Section 4.2).

The focus of our work has been to add delegation capabilities to these relationship-based protocols. *Delegation* occurs when a user allows a service, an application, or another person (a *delegate*) to use the privileges the user has been granted by an RP. Delegation is most useful when a delegate needs to use privileges that the RP has not granted him. Delegation makes a security system's evaluation of delegates' trustworthiness more flexible by allowing trusted users to determine which privileges should be granted to delegates. This increased flexibility is so important to users that they often create insecure ad-hoc techniques to work around security policies that restrict their ability to delegate.

2.2 Security

Delegation systems are *secure* when only users' authorized delegates have access to the delegated user privileges. In online systems, this security is based on the following:

- The set of privileges desired by the user or delegate must be identified.
- The intent of the user to delegate those privileges must be verified.
- The delegate must satisfy delegation criteria before exercising the user's privileges.

The delegation criteria for current authorization systems almost always include authentication of the user or delegate requesting access. To satisfy this criterion, users provide an identifier and then use some associated secret to prove "ownership" of that identifier. The secret information might be a password, a private key, or a secret established via some handshaking protocol. In OpenID, for example, the secret used to verify a user's identifier claim is generated via a Diffie-Helman key exchange during "associate" transaction between an RP and the user's IdP, and unlocked by the user's authentication to the IdP; its exclusivity to the user's identifier is based on the user's unique OpenID URI and an honest DNS [9].

2.3 Selectivity

In many ad-hoc delegation systems, delegates are able to exercise exactly the same set of privileges available to delegating users. This *full delegation* is simple to implement, and it can be very useful when users have a strong trust relationship with their delegates. In other situations, the lack of privilege restrictions in full delegation is hazardous because misbehaving or inept delegates may take actions the delegating user would not approve. For example, web applications users' delegates may have the ability to highjack user accounts, change personalization settings, or permanently delete user data.

One common form of ad-hoc full delegation is *password sharing*, in which users simply give their secret authentication credentials to others. Password sharing is uniformly condemned by security researchers because of the risks of full delegation, but it is often the only delegation mechanism available to users. The only widely-used risk mitigation technique has been the use of multiple personas (different identifier/password combinations), to partition user privileges in case of a breach of security. While users are currently willing to trade increased risk for greater flexibility in this way, the use of new single sign-on (SSO) systems decreases the granularity of delegation possible. Users who share OpenID passwords, for example, are in effect granting their delegates full access on *every* site that uses OpenID authentication. The high risk of full delegation in SSO systems motivates the development of more capable delegation mechanisms.

Selective delegation is the delegation of only a subset of a user's privileges to others. Users increase their security when they conform to the "principle of least privilege" [10] and selectively delegate only those privileges delegates need. For example, delegating privileges on at only a single RP is more secure than giving delegates access to SSO credentials that work with any RP. Selectively delegating the various privileges provided by a single RP necessitates more complex transactions between users and RPs; authentication protocols with built-in selective delegation must at a minimum include a communication channel for identification of the subset of user privileges to be delegated. This results in a more complicated identity protocol. One alternative is an increase in the granularity of RP services. For example, separate, selective delegation of the "read" and "send" privileges in an email system may be enabled either by a selective delegation protocol communicating with a single email service, or by using a full delegation mechanism to control access to two separate service endpoints. In this paper, we focus on the built-in selective delegation case, reserving analysis of a high-granularity full delegation system for future work.

2.4 Simplicity

In addition to the "principle of least privilege," Saltzer *et al.* encourage the security design principle of simplicity, both for reduction of implementation errors ("economy of mechanism") and usability ("psychological acceptability") [10]. In keeping with these principles, delegation mechanisms should be simple to implement and minimally impact performance. In addition, the semantics of delegation mechanisms should match "real-world" delegation scenarios as closely as possible, so that users can understand and trust them. Cryptography-centric approaches, which do not semantically mirror human interactions, have proven too difficult for most users to understand, trust, or use [15].

Chapter 3

SimpleAuth

3.1 Metaprotocol

SimpleAuth is a metaprotocol based on previous work [2] extending OpenID to support delegation. It is not a standalone protocol; instead, it is a simple pattern for adding selective delegation support to relationship-based identity systems. Its concrete implementations take the form of extensions or additions to the messages exchanged during online authentication transactions.

At a high level, SimpleAuth requires just four changes to a relationship-based identity system:

- A series of identifiers is used instead of a single identifier.
- Delegators advertise user-specific permissions lists.
- Delegation policy is set by users and enforced by their identity providers.
- Delegate privileges are determined by delegation lists.

The first is that rather than presenting a single identifier for authentication, delegates provide a series of identifiers that represent the chain of users who delegated the desired privileges, starting with a user who has been granted privileges by the RP and ending with the delegate's own identifier. This series of identifiers corresponds to a *delegation chain* of IdPs through which authentication requests and responses will be passed.

Second, an RP advertises a *permissions list*, in which each item in the list represents a role, privilege, or set of privileges available to the user. Whenever possible this information is included in the "outbound" authentication requests to the user's IdP. The advertised permissions list is specific to the RP; applications in the same domain may therefore choose to implement different privilege models. Additionally, RPs may customize the permissions list advertised based on the user's provided identifier; for example, a photo sharing site might choose to provide more delegable permissions (representing additional features) to paying users with "pro" accounts than to users on free trial accounts. Permissions lists correspond to the lists of boolean authorization values typically stored in a user's session object in current web applications.

Third, the authentication protocol funnels authentication requests carrying permissions lists to the user's IdP, where the user has specified *delegation policy* based on these permissions lists. A delegation policy authorizes the IdP to advertise a user-determined subset of the user's privileges to the IdP of a delegate. In effect, the delegating user's IdP acts as an RP, providing a service to delegates that authenticates them to the original RP. IdPs and RPs use the same mechanism to transmit permissions lists; this imposes a simple, uniform interface on user and delegate IdPs. It is this uniform interface that allows delegation chains to contain an arbitrarily long series of delegators.

The final modification to the identity system is the addition of *delegation lists* to "inbound" authentication responses. These are lists of the privileges actually desired by a user or delegate. As the authentication response works its way back along the delegation chain, each IdP checks to make sure that the delegation list it forwards towards the RP is compliant with its user's delegation policy. This privilege enforcement operation is a simple set intersection between the delegation list and the user's permissions list.

3.2 Simplicity and Security

SimpleAuth selective delegation is simple and efficient. Few modifications to the authentication protocol are required. Bandwidth usage is only slightly increased, since only a small site-specific list is transmitted through the chain, just once per site session. Performance impact is minimized by piggybacking the delegation information on authentication requests, avoiding the delays of additional network traffic.

SimpleAuth is also simple for users to understand. The metaprotocol corresponds closely to the common real-world human interaction pattern manifested in school permission slips, doctors' prescriptions, and work and purchase orders. In addition, users will understand user- and site-specific permissions models (enabled by SimpleAuth) better than they would application privileges expressed using a fixed set of permissions like UNIX's read, write and execute permissions.

SimpleAuth also fulfills the requirements mentioned in Section 2.2 for a secure delegation system. The set of privileges desired is explicitly represented by the delegation list returned to the RP. The delegating users' intent is manifest in the delegation policy that allows the users' IdPs to pass authentication messages along the delegation chain. Finally, the authorization criteria in SimpleAuth are fixed: delegates must verify their identities, appear in the user's predefined delegation policy, and request privileges allowed by that policy. The first criterion is fulfilled by the authentication request/response carrying the delegation information. The second and third criteria are fulfilled when the user's IdP verifies the applicable policy information before it forwards the delegate's authentication response and delegation list up the chain.

3.3 Approach

Policy-based access control in network design has been widely described in terms of four necessary components: a policy management tool, policy repository, and policy decision point (typically bound together as a *policy engine*), and a policy enforcement point [13]. This same taxonomy applies to any access control method based on policies. Since SimpleAuth encourages RPs to maintain a site-specific permissions model (which requires a customized policy enforcement point), it is tempting to suggest that the application should likewise provide the other three components. Indeed, recent delegation-enabled web applications like Google Documents[1] implement *all* the components of policy-based access control within a single application. This approach gives RPs greater control over how users make delegation decisions and may allow an RP to guarantee the performance and reliability of the policy engine.

While RPs may desire this high level of control, there are significant disadvantages to developing and maintaining an internal delegation solution. When an RP implements a delegation mechanism, the RP must either accept delegates' single sign-on credentials or else provision local accounts for delegates. If local accounts are used, even delegates who use SSO systems are forced into managing multiple site-specific identifiers. If the RP accepts SSO logins, a local "handle" for persistent delegation policy must still be established. This increases the user management burden on the RP, decreasing scalability, and means that revoking a delegate's privileges on many RPs necessitates a separate user interaction with each RP. Spreading delegation and revocation functionality across many RPs forces users to learn multiple user interfaces; even requiring users remember to manage delegation information with multiple RPs unacceptably increases users' cognitive burden. A plurality of delegation interfaces also makes users more vulnerable to phishing attacks.

Externalizing the policy engine, as in SimpleAuth, addresses these issues. Pushing the policy decision point out to user IdPs allows RPs to store only privileged users' permissions models, and not delegates'. No site-specific identifiers, credentials,

[1] `http://docs.google.com`

or user interfaces are required. Users can delegate or revoke a delegate's privileges on many RPs simultaneously.

More importantly, SimpleAuth gives responsibility for implementing delegation to identity providers, who have a strong economic incentive to differentiate their services with added functionality and refined usability. Delegation is important as a component of a successful user-centric identity system, but it is not part of the "core competency" of most RPs. Augmenting a widely-used authentication protocol with selective delegation allows this feature to be used even with RPs whose implementors do not have the resources or expertise to create an effective delegation system.

3.4 Validation

Relationship-based identity systems by definition share a single trait: their reliance on authentication-time communication between an RP and a user's IdP. Accordingly, any metaprotocol applicable to relationship-based identity systems in general must rely solely upon this property. SimpleAuth therefore cannot be used "out of the box" with *all* relationship-based protocols because it requires a *two-way* means of communicating delegation information; indeed, it is trivial to modify a relationship-based protocol to be incompatible with SimpleAuth by blocking the communication of any information besides authentication messages between RPs and IdPs. We believe, however, that it is similarly simple to modify any relationship-based protocol to *support* SimpleAuth by enabling this communication channel.

To verify that SimpleAuth generalizes across multiple authentication protocols, we have implemented a SimpleAuth extension to the Surrogate Secure Remote Password protocol (sSRP). The SimplePermissions OpenID extension [2] is equivalent to a SimpleAuth implementation for OpenID. Because OpenID and sSRP have such different communication flows, as shown in Figure 3.1, these SimpleAuth im-

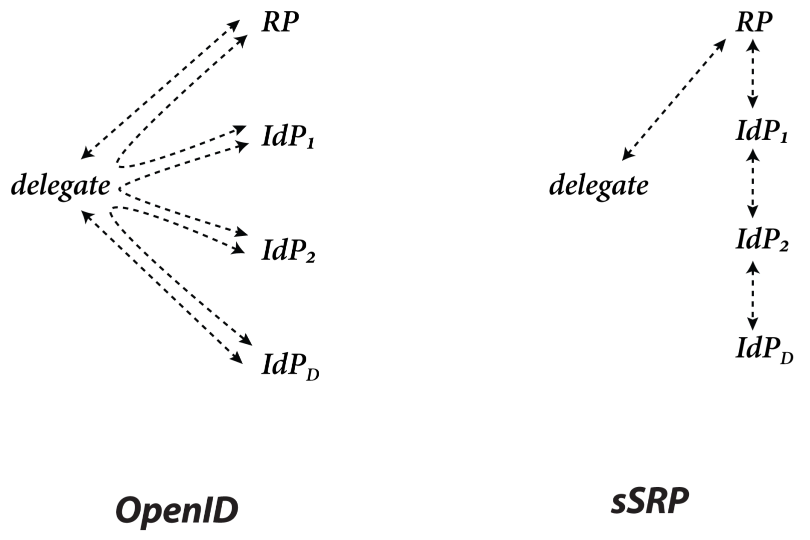

Figure 3.1: Differing OpenID and sSRP message flows

plementations, considered together, are significant evidence that the metaprotocol is applicable to relationship-based authentication protocols in general.

Chapter 4

SimpleAuth Implementations

4.1 OpenID SimpleAuth implementation

OpenID authentication protocol

OpenID [9] is a relationship-based identity system designed originally to provide SSO capabilities to weblogs. When a user submits his identifier (a URL or XRI) to an RP, the RP determines the URL of the user's IdP through a process called Yadis discovery. The user's browser is then redirected to IdP with a `checkid_setup` or `checkid_immediate` message. The user then logs in to the IdP, using whatever authentication method the IdP requires. After authentication, the user is redirected to the RP with a signed `id_res` message confirming the user's claimed identifier. The signature is checked using either a session key previously established in an `associate` message, or a confirming `check_authentication` message.

SimpleAuth modifications to OpenID

SimplePermissions [2], a selective delegation extension to OpenID, was the predecessor of SimpleAuth and is equivalent to a OpenID-based SimpleAuth implementation. A typical use of the modified protocol is shown in Figure 4.1. The OpenID protocol has been enhanced with the four SimpleAuth modifications:

- The delegate's client submits the OpenID URL of the first user in the delegation chain to the RP. When the client is redirected to that user's IdP, it submits

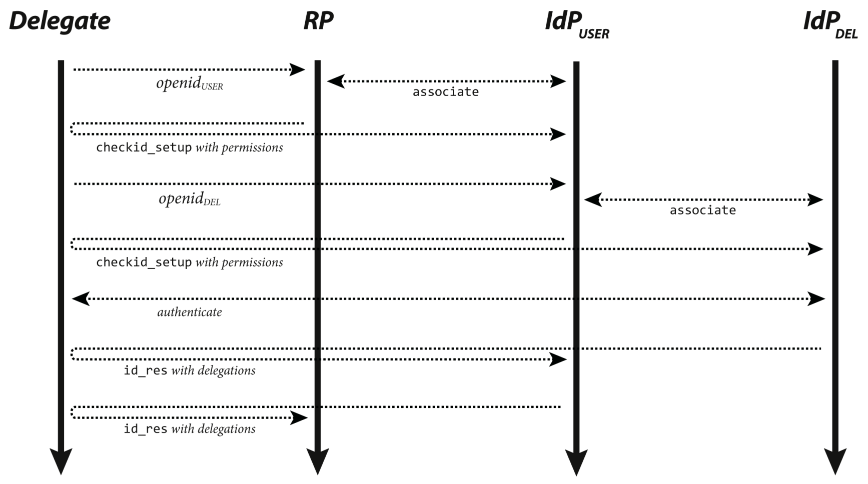

Figure 4.1: OpenID with SimpleAuth extensions

the next OpenID URL in the chain, and so on, until it authenticates to the delegate's own IdP.

- Permissions lists are included in each `checkid_immediate` or `checkid_setup` message. The privileges included in the advertised permissions list at each stage are based on the delegation policy the IdP's user has established for the delegate in question. Privileges are expressed as extension attributes in the `openid.permissions.*` namespace.
- A delegation list is returned from the IdPs with each `id_res` message. When an IdP receives an `id_res` response, the included set of permissions is intersected with the set of permissions allowed by the user's delegation policy, to prevent privilege escalation attacks.
- Creation of delegation policy is not shown in the figure, since the specific syntax, format, and semantics used to express the user's policy are outside the scope of the SimpleAuth metaprotocol.

Note that the OpenID protocol itself is unchanged (adding attributes to OpenID messages is allowed by the protocol); the changes are all made to the IdP and the delegate client.

4.2 sSRP authentication protocol

The Surrogate Secure Remote Password protocol (sSRP) is a relationship-based authentication protocol developed for use in wireless internet access points [5]. It is based on the Secure Password Protocol (SRP) [19, 17], which has been standardized as RFCs 2945 [16] and 5054 [18]. SRP is a zero-knowledge proof-of-knowledge (ZKPK) protocol, meaning that the user's secret, a password, is never shared with anyone; even the entity authenticating the user receives only a "password verifier" based on a salted hash of the password. SRP has several desirable properties:

- SRP's ZKPK approach means that no information useful to attackers is ever transmitted across the network, making SRP effectively immune to several classes of attacks that have been problematic in other protocols. For example, the session secret created during authentication is not vulnerable to "man in the middle" attacks, so SRP authentication messages can be safely proxied through untrusted third parties.
- SRP is resistant to phishing attacks, because the user's password is never sent across the network. In addition, SRP authentication allows mutual authentication; the user can confirm the identity of the authenticating party based on its posession of the correct password verifier.
- During authentication transactions SRP uses the minimum number of messages necessary to provide replay prevention and mutual authentication (one session parameter exchange and one authentication exchange per authentication).

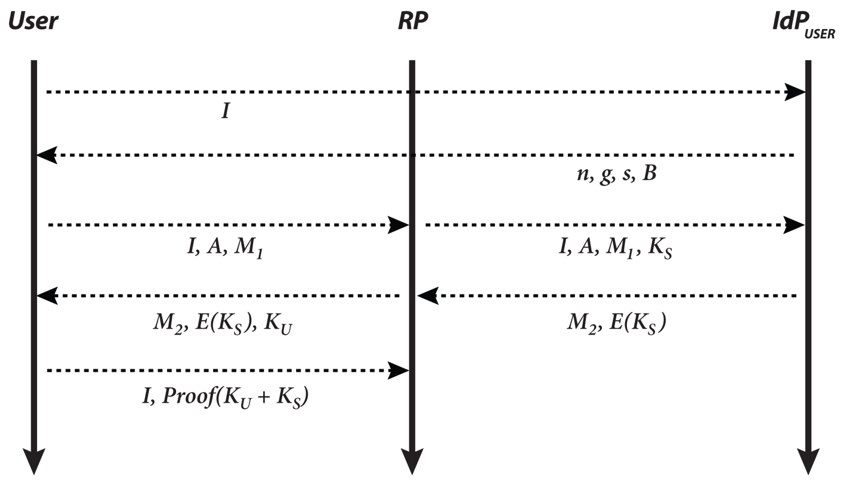

Figure 4.2: Basic sSRP protocol

sSRP is a modification of the original SRP protocol. Unlike SRP (which requires the RP to authenticate users with a password verifier), sSRP is a relationship-based system that allows users to authenticate to RPs using an identity provider. This is done by making the RP a proxy between a user and his identity provider, and adding token splitting to verify that authentication has taken place. Using the RP as a proxy is safe because no secret information is exposed to *any* "man in the middle" during authentication.

Just as in SRP, sSRP has two distinct phases (shown in Figure 4.2). In the first phase, the user submits his identifier to the RP. User identifiers are globally unique and *discoverable*, meaning that the identifier can be used to find the location of the user's IdP. Examples of discoverable identifiers include URLs and XRIs [14] in OpenID, or email addresses as in SAW [12]. After the RP finds the user IdP's sSRP endpoint, it passes on the unmodified identifier to the user's IdP. The response includes the large prime n, safe root g, and salt s associated with the IdP's password verifier, as well as a public message B based on a new random session secret b. The RP forwards the parameters unchanged to the user.

In the authentication phase of the protocol, the user client chooses its own ephemeral secret a and creates a public message A from it. Only someone with access to A, B, and the correct password verifier (based on the password, n, g, and s) can compute the new shared session secret S. To prove possession of the password, the user sends A along with a message M_1 signed with S to the RP.

When the RP receives A and M_1, it generates a session key for the user. This key is split into two halves, K_U and K_S. K_S is sent along with M_1 and A to the user's IdP. The IdP uses A to create S, which it uses to verify M_1. If verification succeeds, the IdP uses S to sign M_2 and to encrypt K_S, which are returned to the RP.

The RP adds K_U to the authentication response and forwards it on to the user. The user verifies M_2 (if mutual authentication is desired) and then uses S to decrypt K_S. Finally, the user submits a proof of its possession of K_U *and* K_S to the RP. Both halves of the session key are necessary to prove that authentication has occurred.

4.3 SimpleAuth modifications to sSRP

SimpleAuth is a generalization of earlier work using OpenID authentication [2]. Since a working OpenI-based implementation of SimpleAuth already exists, the primary purpose of our work has been the extension of the metaprotocol to handle other relationship-based systems. sSRP was chosen as for our proof-of-generality implementation not only because of its desirable security features, but also because its message flow differs significantly from OpenID's. If SimpleAuth extensions can be made to protocols as different as OpenID and sSRP, it argues for the applicability of SimpleAuth to *all* relationship-based authentication methods.

A SimpleAuth-augmented version of the sSRP protocol is shown in Figure 4.3. The basic sSRP protocol has been enhanced with the four SimpleAuth modifications:

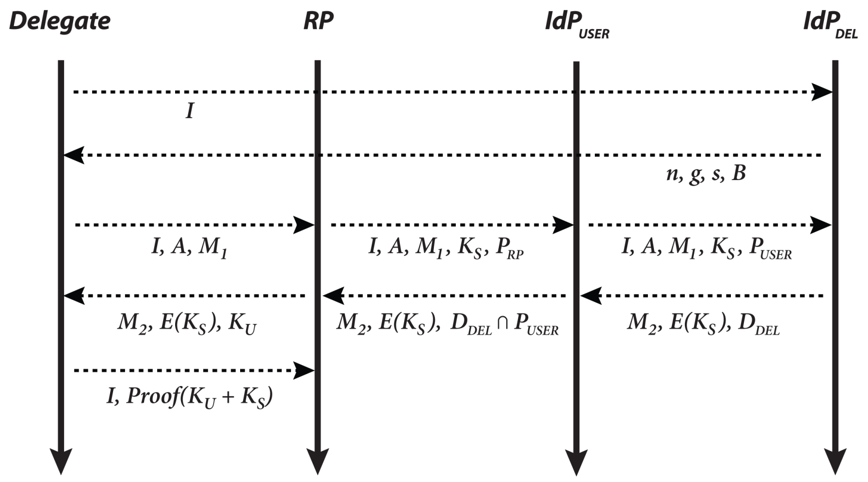

Figure 4.3: sSRP with SimpleAuth extensions

- The user submits a chain of identifiers, I, instead of a single identifier. At each step in the delegation chain, the RP or IdP involved dynamically discovers the sSRP endpoint of the next IdP in the chain.
- Permissions lists are sent from the RP through each IdP in the delegation chain. The privileges included in the advertised permissions list at each stage are based on the delegation policy the IdP's user has established for the delegate in question.
- Delegation lists are collected from the IdPs during the return leg of the authentication phase. At each stage, the set of permissions requested by a delegate is intersected with the set of permissions allowed by the user's delegation policy, to prevent privilege escalation attacks.
- Creation of delegation policy is not shown in the figure, since the specific syntax, format, and semantics used to express the user's policy are outside the scope of the SimpleAuth metaprotocol.

4.4 sSRP-specific protocol issues

In this section we examine the modifications to the sSRP protocol made during our implementation which are not a part of the generalized SimpleAuth metaprotocol.

Discoverable identifiers

Rather than create a new identifier scheme, our sSRP-based SimpleAuth implementation uses a pluggable module to allow use of several types of discoverable user identifier. Discoverable identifiers like OpenID URLs and email addresses are easy for users to remember and type. Also, while the SimpleAuth pattern does not mandate that user identifiers be discoverable, requiring discoverability greatly simplifies implementation of the RP and IDP SimpleAuth modules: authentication messages include the delegation chain as a simple list of identifiers instead of carefully maintaining separate message routing information, and because the identifiers are public information, delegation chains of identifiers need not necessarily be stored or transmitted securely.

Additionally, identifiers with an indirect discovery process protect users from identity provider lock-in (which was one the major drawbacks of previous attempts at internet-scale identity systems including Microsoft Passport). In OpenID, this is done by separating a user's "ownership" of an identifier from responsibility for authenticating that user: instead of pointing directly at an authentication service, a user's OpenID URL points to a *pointer* to the user's chosen IdP. The OpenID community refers to this technique, confusingly, as "delegation."

This outsourcing of authentication responsibilities is important for two reasons. First, it is unrealistic to require each user of an internet-scale identity system to run their own IdP. This technique allows authentication services to be provided by knowledgeable security professionals, rather than untrained users. Secondly, it provides a users some protection against misbehaving IdPs: if an identity provider abuses its trusted position, users can point their identifiers at a more trustworthy IdP. For both these reasons, and for the user's convenience when submitting the delegation chain, we require that each identifier in the chain be globally unique and discoverable. Note that we do not require that the identifiers share the same local namespace: theoretically, a delegation chain might consist of a mix of email addresses, XRIs, OpenIDs, or any other discoverable identifier type.

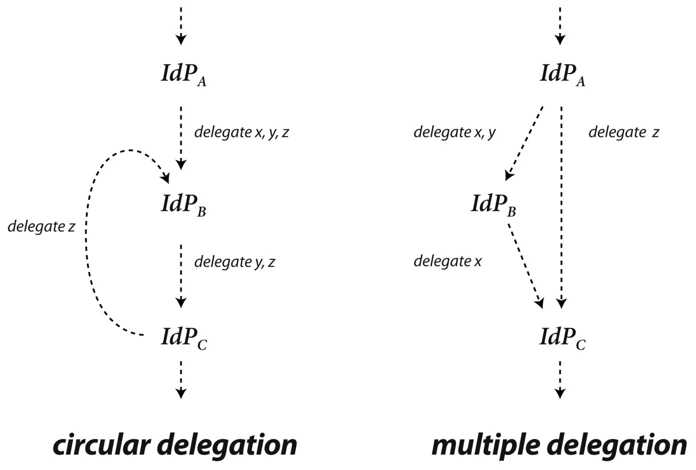

Figure 4.4: Problematic delegation chains

Shifting identifiers

While implementing the protocol, several identifier chain management issues became apparent, as shown in Figure 4.4. An IdP in the delegation chain must know where in the chain it is, so that it can correctly send sSRP messages to the next IdP in line. Although it is unlikely, nothing prevents the creation of a *circular delegation* - that is, a delegation chain in which Alice is delegated privileges by Bob which were originally delegated to him by Alice. When Alice authenticates, how can Alice's IdP determine whether to return a delegation list to Bob's IdP or to the RP? A more common situation might be *multiple delegation*, in which a person is delegated privileges by two different users (or series of users) who are themselves both delegates of a single user. The problem corresponds to the choice among multiple paths between two nodes in a directed graph: which path should the delegate's IdP take?

While this problem was never satisfactorily resolved in our OpenID implementation, the provision of the entire delegation list to each of the intermediate IdPs in the sSRP version provides a convenient solution. The identifiers of the delegation chain are actually kept in two lists, one representing the logical delegate and the other representing the logical delegator. For example, if Alice has delegated her privileges at "foo.com" to Bob, who has further delegated to Charles, the delegator would be `foo.com` while the delegate would be `Alice > Bob > Charles` (`>` is used because it does not appear in valid urls or email addresses). When Alice's IdP receives a message with these identifier strings, it *shifts* the identifiers; the delegator becomes `foo.com > Alice` while the delegate is `Bob > Charles`. This makes the correct route through the delegation "graph" unambiguous. For example, Charles's IdP can easily differentiate between delegators `foo.com > Alice` and `foo.com > Alice > Bob`, even if both Alice and Bob have delegated him Alice's privileges.

Delegation policy features

One of the four key components of SimpleAuth is the delegation policy that users create at their IdPs. However, the SimpleAuh metaprotocol intentionally does not define the form that these policies take, or the interface by which users create them. While simple delegation policies may be simply a delegate identifier paired with a list of permissions, policies could also be expressed with a rule or policy language like XACML [7], be based on external data sources such as reputation systems, or could be otherwise computed rather than set explicitly by the user for each delegate.

One important advantage of relataionship-based systems over credential-based systems is that revocation is simple. In credential-based systems, the canonical solution to the revocation problem is the creation and frequent consultation of revocation lists: in effect, this negates the protocols' offline capabilities. IdPs in SimpleAuth-enabled systems can revoke delegates' privileges at any time (at least at session granularity) by refusing to authenticate them.

An interesting consequence is that it becomes trivial to put time or usage limits on delegations. For example, a single time stamp or usage counter can be used in a delegation policy record to limit use of that delegation to a specific range of dates, a specific time of the day, or to a maximum number of uses (including one-time-use delegation). These meta-restrictions are simply additional features an IdP may choose to provide, and are outside the scope of the SimpleAuth protocol.

Additional restrictions may be built into delegation policies when SimpleAuth is implemented in a way that exposes the entire delegation chain to each IdP in the chain. For example, a user may decide to give privileges to a delegate but not allow them to further delegate the permissions, or to forbid delegates authenticated by identity providers the user distrusts. Since our implementation is a prototype system designed only to demonstrate the SimpleAuth metaprotocol, delegation policy is managed as a set of `(user, delegate, privilege)` tuples, with none of these more advanced features.

Defense against delegate hopping

One problem unique to multi-step delegation is that a malicious IdP could try to skip over its predecessors in the delegation chain, as shown in Figure 4.5. This is only possible in systems like our sSRP implementation, where each IdP sees the entire delegation chain but intermediate IdPs' modifications to the message do not affect the authentication of the delegate (only the delegate's authorization). This vulnerability would occur only in multi-step delegation, and would only allow privilege escalation by partially-privileged delegates running their own IdPs.

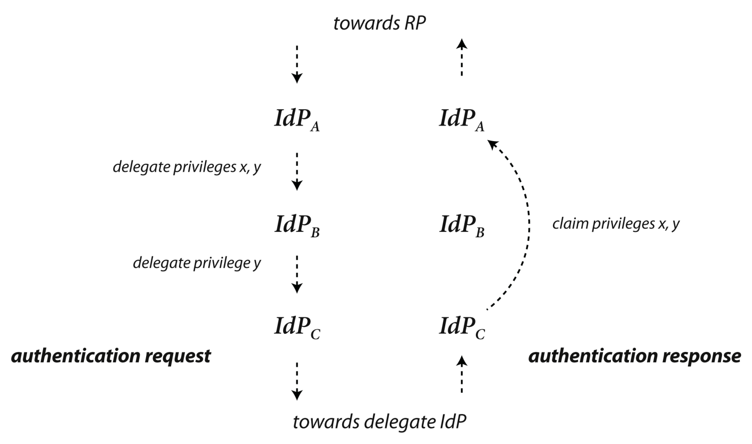

Figure 4.5: Delegate-hopping attack

To prevent this attack, each identity provider in the delegation chain needs assurance that the IdP returning a delegation list is the same IdP to which it previously sent the user's permissions list, i.e. that the delegation chain has been followed, step-by-step, in both directions. In our implementation, this is achieved by appending a random handle to every sSRP message. When an IdP receives a request-type message, it stores the message handle and new random handle locally, and replaces the handle in the message with its new handle. On receipt of a response-type message, the IdP checks to make sure that the message handle matches the message handle sent before processing, and includes the original message handle it received in its response. In a TLS-encrypted transaction, this gives very strong assurance that the responding IdP is not an impersonator. Even using non-confidential messages, however, it provides reasonably strong protection, since it is unlikely that first, that a malicious user would be delegated to initially, and secondly, that such a malicious user would have access to the message traffic (and thus, the appropriate handle) between a pair of arbitrary internet hosts (the other IdPs).

Asynchrony

For scalability reasons, it is desirable to make the protocol as asynchronous and stateless as possible. Together, these two properties minimize the negative impact of sending multiple messages over the network during each authentication transaction. This goal meshes nicely with the delegate-hopping prevention handles mentioned previously in this section. Because each hop of each message already has a unique handle, message handles can be used as keys to encapsulate all of a message's state in an associative datastore. When a mid-chain IdP receives a SimpleAuth message, it simply retrieves the message state, consults the appropriate policy, and sends the message on. The simplicity of the SimpleAuth IdP module is important because a simple implementation can be inspected for flaws and tuned for performance. IdPs compete not only on features but on security and performance, and any new feature that detracts too greatly from the IdP's efficiency or auditability will face significant barriers to adoption.

4.5 sSRP SimpleAuth implementation

We wrote our proof-of-concept implementation in Python as a single module containing `Client`, `IdPServer` and `RPServer` classes. These classes provide an external API, so that they can be embedded for internal use by existing web applications. Additionally, each class can run as a server (based on the Python standard library's `BaseHTTPServer`) during testing and development.

We also created a WebKit[1]-based demonstration browser using the PyObjC[2] bridge to connect to the SimpleAuth `Client` class. The browser features a "Login" button as an anti-phishing measure (as described in Section 4.6), and automatic detection of SimpleAuth-enabled sites' login endpoints by detecting HTML `<form>`

[1]`http://webkit.org`
[2]`http://pyobjc.sourceforge.net/`

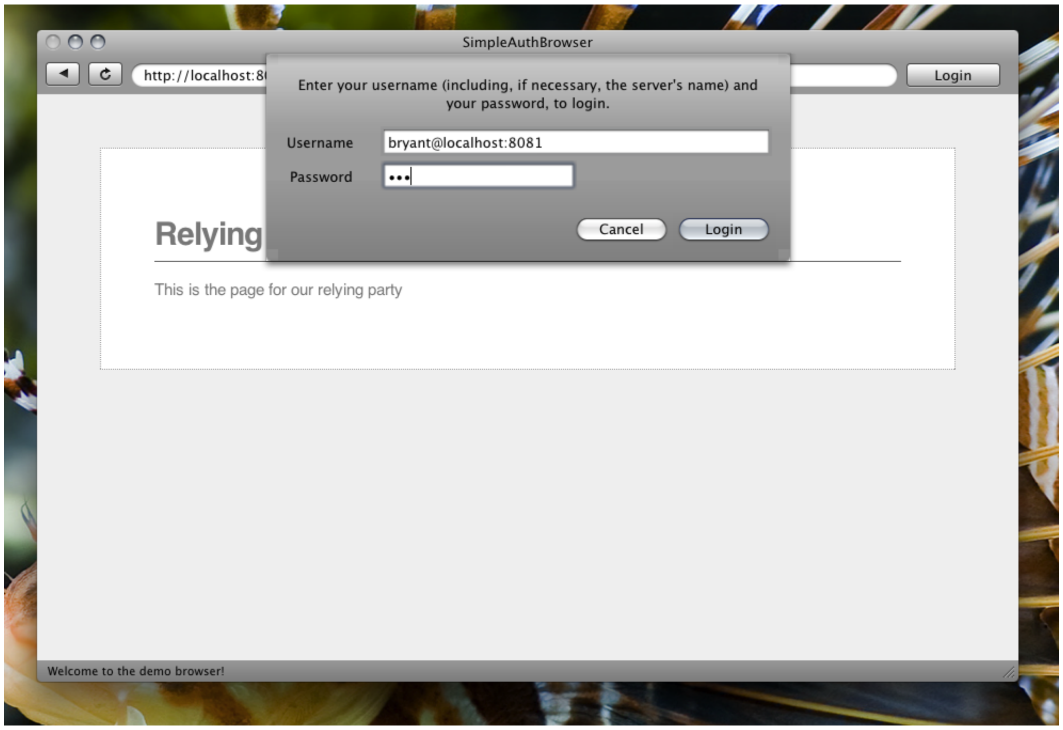
SimpleAuthBrowser
http://localhost:8
Login
Enter your username (including, if necessary, the server's name) and your password, to login.
Username
bryant@localhost:8081
Password
•••
Cancel
Login
Relying
This is the page for our relying party
Welcome to the demo browser!

Figure 4.6: Demonstration SimpleAuth-enabled browser

tags marked up with the "simpleauth-login" class. On SimpleAuth enabled sites, the login button activates, allowing the user to enter an identifier and password, as shown in Figure 4.6. Acceptable identifiers include OpenID URLs and pseudo-email addresses (`username@idp`).

Our final source code, including the PyObjC UI delegate, command line client, and the IdP and RP web servers with demonstration web applications (including HTML and CSS) is only nine hundred lines of Python script, and was written without prior knowledge of Objective C or any of the various Python libraries involved. This relative ease of implementation is strong evidence of the SimpleAuth pattern's simplicity, and demonstrates that this technology could be easily integrated into browser and web application software.

4.6 sSRP threat analysis

Direct attacks

Direct attacks are those that attempt to computationally "crack" a user's credentials. In sSRP, a user's credential is his password. Because no information that can be used to derive the password is ever transmitted across the network, sSRP passwords are immune from direct attacks by untrusted third parties. The only party that can attempt an offline attack to compromise a user's credential is that user's IdP, and only then by means of a dictionary-based attempts to find a hash collision. Attacks against user password verifiers in this way are unlikely; in a relationship-based protocol the IdP is already trusted to assert its users' identities, and must therefore be trusted both to keep users' password verifiers secret and not to attack their own users' credentials.

Phishing and man-in-the-middle attacks

Phishing is the technique of coercing users to give up authentication credentials by presenting them with a fraudulent authentication interface; in the case of online delegation, this is done by impersonating a legitimate RP or IdP. sSRP is exceptional among relationship-based protocols in its intrinsic resistance to phishing attacks because users' authentication credentials are never transmitted over the network. Imposter RPs and IdPs will never receive the user's authentication credentials, even if they display a convincing facade to users. This holds for eavesdropping third parties ("men in the middle") as well: none of the information exchanged over the network during an sSRP authentication transaction can feasibly be used by attackers to compromise either a user's password or the session secret the user's client establishes with the IdP.

The one place that users' passwords are potentially at risk is the sSRP client itself; since an insecure sSRP client implementation could compromise the user's SSO credential, care must be taken to prevent attackers from impersonating the client's interface. An in-page client could theoretically be built as a signed applet, although such an implementation would rely on users' understanding and verification of the applet's signature for its security. A better option would be to implement the sSRP client outside the page ("in chrome") as a browser extension or operating system module. These methods would allow user interfaces not easily duplicated by attackers who have not already compromised the user's system.

sSRP is also resistant to phishing for non-credential information. Users set up policy exclusively at their identity providers, and therefore can't be easily tricked into transmitting this information to third parties. Users convey no secret information to their delegates except for the fact that a delegation has occured - the chain of public identifiers delegates will use to access user privileges can be safely communicated even over insecure channels. Phishers attempting to pose as a user's IdP are foiled; the IdP directly authenticates users with the SRP protocol, which provides mutual authentication.

Pharming and policy leaks

Pharming is the technique of compromising a system by controlling the information it receives from the domain name system (DNS). When used in conjunction with phishing attacks, pharming allows attackers to impersonate a website, displaying a trusted URL to web users while delivering compromised pages. As an enabler of man-in-the-middle attacks, pharming allows attackers to breach security when a connection to a specific domain (for example, an identity provider) is required.

In our sSRP implementation of SimpleAuth, the authentication process is secure against pharming of the user's client because the base sSRP protocol is intrinsically resistant to man-in-the-middle attacks and phishing. However, SimpleAuth (like any relationship-based web authentication protocol) relies on an honest DNS for security. If an RP is pharmed, it will forward authentication requests to phony identity providers. This is a problem for any relationship-based protocol, not just those with delegation enabled, and is outside the scope of SimpleAuth. It is clear, however, that allowing authentication transactions only over SSL connections with verified certificates would protect against most pharming attacks. While requiring web users to correctly verify SSL certificates is unrealistic, RPs may reasonably be expected to do so.

While sSRP authentication credentials are safe even from pharmers, our sSRP SimpleAuth implementation does expose one piece of user data to attack: the user's delegation policy. User policy may contain confidential information, including the roles or privilege levels assigned to delegates, which could be used in other attacks. For example, discovering delegates' roles might allow an attacker to formulate a more effective social engineering attack. Since a user's policy for a delegate is only ever transmitted to that delegate's IdP, this information is not casually available. In the case of pharming, however, attackers could fool an IdP into disclosing this sensitive information by masquerading as both the upstream service and the delegate's IdP. Just as RPs protect the authentication protocol by verifying IdP SSL certificates, IdPs can protect the delegation policy by using only checked-certificate SSL connections to communicate with downstream IdPs; correct implementation of SSL is a realistic requirement for IdPs.

Privilege escalation

Privilege escalation attacks are attempts by malicious but valid delegates to use more privileges than a user or upstream delegate has granted them. In our sSRP SimpleAuth implementation, a delegate may attempt to gain privileges by modifying the delegation lists returned by his IdP. A naïve attempt to add permissions to the delegation list will fail because the list is inspected by the delegating user's identity provider before being sent upstream, towards the RP. Any permissions asserted by a delegate that are not contained in the user's delegation policy are removed.

A more subtle attack is delegate hopping, as described in Section 4.4. Delegates several steps removed from the original delegating user may attempt to go "over the head" of the delegates upstream from them, using a subset of the user's privileges approved by the user but not by upstream delegates. To defend against this attack, the protocol must enforce the traversal of the entire delegation chain during the trip from the authenticating delegate's IdP to the RP. In our implementation, this is done by having IdPs at each downward step in the delegation chain create a non-guessable key to verify that the same route has been traversed coming back up. These hop-specific keys should be protected from delegate eavesdroppers using an SSL connection. In a synchronous SimpleAuth implementation, SSL connections would provide all the necessary request origin assurance necessary - asynchronous implementations must either use client-side SSL at each identity provider or implement an application-layer method as we have.

Sidejacking

At the end of an sSRP SimpleAuth transaction, the authenticating user or delegate has created shared secrets with both the RP and his IdP. How these secrets are used to maintain web application state or control access to RP resources is outside the scope of authentication and delegation protocols. In particular, users of secure techniques for authentication and delegation may still be vulnerable to "sidejacking" attacks, when attackers wait until after a user authenticates to step in and take over their session. RP and IdP implementors should use techniques like SessionLock [3] to maintain security after authentication.

Chapter 5

Related Work

5.1 Authorization-based access control

SimpleAuth is similar in several ways to authorization-based access control (ABAC) as proposed by Alan Karp [6]. Just as in SimplePermissions, ABAC systems decouple the authorization policy engine from an RP's services. ABAC's trust model is similar to SimpleAuth's: the relying party's trust relationship is with a trusted service or IdP, rather than with the user himself (in the case of ABAC, this is the signer of the user's SAML assertions). Karp asserts that ABAC is more scalable, evolvable, private, manageable, and secure than role-based access control because of site-specific permission models and easy delegation; these assertions apply equally to SimpleAuth.

ABAC does differ from SimpleAuth in significant ways. At its core, ABAC is a capabilities system, which means that the "authorizations" (capability tokens, in Karp's paper) created for a user are not tied to a delegation but allow anyone in possession of an authorization token to access user privileges. This makes anonymous delegate use cases possible, but also makes authorizations vulnerable to theft. Thus, the authorizations must be stored and transported securely. Because ABAC authorizations are as difficult to revoke as any other credential, user security is dependent on careful doling out of authorizations; this makes maintaining a persistent delegation more complicated than in SimpleAuth.

A final way that ABAC differs from SimpleAuth is in the user interface for delegation. Using ABAC, a user interacts directly with an RP, retrieving authorizations. The user is then responsible for conveying these tokens to delegates in a secure manner. Because SimpleAuth policies are stored within a user's identity provider and transmitted automatically during authentication transactions, users do not need to manually store or transmit any sensitive credential or policy tokens. A delegating user interacts only with his identity provider, using his IdP's single consistent interface to set policy.

5.2 Certificate-based "limited delegation"

Researchers at Dartmouth developed a technique called "limited delegation for client-side SSL" [11]. It is similar to SimpleAuth in that a delegation-enabled web site creates a site-specific permissions model, described as a set of boolean attributes that each represent a privilege or set of privileges. Users choose the subset of the available privileges to be granted to a delegate. When a delegate attempts use the delegated privileges, they must first authenticate using their own credentials.

Unlike SimpleAuth, the "limited delegation" technique is built on a credential-based system, client-side SSL authentication. The system requires an PKI-based identity system, in which users manage public and private keys and client-side SSL certificates. The delegation process involves creation of a special certificate, signed with the delegating user's private key, which includes the public key of the delegate along with the list of delegated privileges.

The use of SSL certificates as both delegation tokens and delegate credentials provides very strong guarantees about the identities and intentions of users and delegates. The client-side certificates must be stored locally and managed via a browser, so user and delegate identities are not easily portable across devices. While this problem is common to other certificate-based and hybrid approaches, including Identity

Metasystem [8], the use of delegation certificates further complicates the credential synchronization problem. Because users do not maintain online revocation lists, the delegation certificates must be time-limited; this may create either a security problem (misbehaving delegates whose credentials take too long to expire) or a usability burden (frequent creation of replacement delegation certificates).

The technique's main limitation is the reliance on HTTP cookies for permissions model discovery. This means that users who do not accept cookies (to avoid online advertising tracking, for example) are unable to delegate their privileges. Because the cookies are sent to the browser before authentication, RPs cannot advertise user-specific permissions models. Also, because permission models are discovered while visiting a site, rather than during authentication, the user's browser must accept and store the permissions model for *every* site visited, even those on which the user has no privileges. This makes the delegation method a new spam vector, since cookies (which are normally be hidden from users) must be displayed in the permissions management interface.

5.3 OAuth

The OAuth protocol is a generic methodology for API authentication, unique among the protocols mentioned here because it is gaining widespread adoption [1]. Just like SimpleAuth, OAuth is not tied specifically to any one authentication method. OAuth was developed specifically for the "mashup" use case, in which a user delegates to a service (a "consumer"), giving it access to user data stored by other services ("service providers"). The OAuth specification defines the format of the authorization and access tokens exchanged by users and service providers.

While SimpleAuth is designed for both user-to-user and user-to-service delegation, OAuth has been tuned specifically for use by consumer services. This means that there is no provision in the specification for end users to store or manage authorization tokens. The delegation process involves consumers redirecting delegating users to service providers for authorization decisions; users are then redirected back to the consumer, carrying notification of authorization for the consumer.

Although this approach has the advantage of being somewhat interoperable with earlier systems (AOL's OpenAuth[1], Yahoo!'s BBAuth[2], and Google's AuthSub[3]), increasing the likelihood of broad adoption, it means that there will be no standard user interface for delegation: each service provider must provide its own interface. Users must similarly visit each service provider separately to revoke consumer access.

A bigger problem with the OAuth approach is the requirement that the consumer establish a relationship with the service providers in advance of the delegation. This relationship is typically realized when a service provider grants the consumer an API key. This makes OAuth unsuitable for dynamic delegation decisions, in which the consumer has no preexisting relationship with the service provider. When SimpleAuth is paired with the proper authentication system, it allows full delegation even on non-cooperating RPs; OAuth features no such capability.

[1]`http://dev.aol.com/openauth`
[2]`http://developer.yahoo.com/auth/`
[3]`http://code.google.com/apis/accounts/docs/AuthForWebApps.html`

Chapter 6

Conclusion

Delegation is an essential feature of human identity systems. As people increasingly participate in online identity systems, it is important that we give users the ability to selectively delegate their privileges to others. The ability to selectively delegate increases the security of users by allowing conformance to the principle of least privilege. Building delegation into identity systems also increases security by decreasing the likelihood of ad-hoc delegation via shared credentials, a technique that becomes riskier as more web users use single sign-on systems.

SimpleAuth is a pattern for adding this selective delegation to relationship-based protocols. In the SimpleAuth model, relying parties delegate privileges to users, who may further delegate privileges to delegates. These delegated privileges are advertised as a *permissions list* during authentication; this allows relying parties to expose user-specific permissions models. The permissions list is passed down a delegation chain of identity providers, modified at each step by users' *delegation policy*. After a delegate authenticates to their own identity provider, that identity provider passes a *delegation list* back up the delegation chain to the relying party. At each step up the chain, a delegation policy is enforced, by passing along assertions for only those privileges the delegator wants to give the delegate.

SimpleAuth was first implemented as an extension to OpenID. To prove the generality of the SimpleAuth metaprotocol, we have built a SimpleAuth implementation based on the sSRP protocol. The ease of implementing SimpleAuth on two relationship-based authentication protocols with very different message flows is strong evidence for the technique's generality. All SimpleAuth requires of the underlying authentication protocol is a safe two-way communication channel between relying parties and identity providers, to convey permissions and delegation lists.

The SimpleAuth pattern does not compromise the security of the underlying authentication protocol. SimpleAuth is also simple, both for users to understand and for developers to implement. SimpleAuth gives relationship-based identity systems greater flexibility and increases the user security.

Bibliography

[1] OAuth Core 1.0 Specification.

[2] SimplePermissions: an OpenID Extension for Delegation and Permissions Model Discovery. Tech. Rep. 200801, Brigham Young University Enterprise Computing Lab, 2008.

[3] Adida, B. SessionLock: Securing Web Sessions against Eavesdropping. In *WWW2008: Proceedings of the Seventeenth World Wide Web Conference* (2008).

[4] Bhargav-Spantzel, A., Camenisch, J., Gross, T., and Sommer, D. User centricity: A taxonomy and open issues. *Journal of Computer Security 15*, 5 (2007), 493–527.

[5] Harding, A., van der Horst, T., and Seamons, K. Wireless Authentication using Remote Passwords. In *1st ACM Conference on Wireless Network Security (WiSec)* (Alexandria, VA, March 2008).

[6] Karp, A. Authorization Based Access Control for the Services Oriented Architecture. *Proceedings of the 4th International Confereence on Creating, Connecting and Collaborating through Computing (C5 2006), Berkeley, CA, IEEE Press, January* (2006), 2006–3.

[7] Moses, T., et al. eXtensible Access Control Markup Language (XACML) Version 2.0. *OASIS Standard 200502* (2005).

[8] Nanda, A. Identity Selector Interoperability Profile.

[9] Recordon, D., Hoyt, J., Hardt, D., Bufu, J., and Fitzpatrick, B. OpenID 2.0 Authentication.

[10] Saltzer, J., and Schroeder, M. The protection of information in computer systems. *IEEE Proceedings 63* (1975), 1278–1308.

[11] Santos, N., and Smith, S. Limited Delegation for Client-Side SSL. *6th Annual PKI R&D Workshop, April* (2007), 17–19.

[12] van der Horst, T. W., and Seamons, K. E. Simple Authentication for the Web. In *3rd International Conference on Security and Privacy in Communication Networks* (Nice, France, September 2007).

[13] Verma, D. Simplifying network administration using policy-based management. *IEEE Network 16*, 2 (2002), 20–26.

[14] Wachob, G., et al. Extensible Resource Identifier (XRI) Syntax and Resolution Specification, 2003.

[15] Whitten, A., and Tygar, J. Why Johnny Can't Encrypt: A Usability Case Study of PGP 5.0. *Proceedings of the 8th USENIX Security Symposium, August* (1999).

[16] Wu, T. The SRP Authentication and Key Exchange System.

[17] Wu, T. SRP-6: Improvements and Refinements to the Secure Remote Password Protocol. *Submission to the IEEE P 1363* (2002).

[18] Wu, T. Using the Secure Remote Password (SRP) Protocol for TLS Authentication.

[19] Wu, T., et al. The Secure Remote Password Protocol. *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium 1* (1998), 97–111.