TRUSTBROKER:
A DEFENSE AGAINST IDENTITY THEFT FROM ONLINE TRANSACTIONS

by

Michael G. Edvalson

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

April 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Michael G. Edvalson

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

Date	Kent E. Seamons, Chair
Date	Charles D. Knutson
Date	Sean C. Warnick

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Michael G. Edvalson 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

Kent E. Seamons
Chair, Graduate Committee

Accepted for the Department

Parris K. Egbert
Graduate Coordinator

Accepted for the College

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

ABSTRACT

TRUSTBROKER:
A DEFENSE AGAINST IDENTITY THEFT FROM ONLINE TRANSACTIONS

Michael G. Edvalson

Department of Computer Science

Master of Science

The proliferation of online services over the years has encouraged more and more people to participate in Internet activities. Many web sites request personal and sensitive information needed to deliver the desired service. Unfortunately, it is difficult to distinguish the sites that can be trusted to protect such information from those that cannot. Many attempts to make the Internet easier to use introduce new security and privacy problems. On the other hand, most attempts at creating a safe online environment produce systems that are cryptic and hard to use. The TrustBroker system is based on a specialized online repository that safely stores user information and helps the user determine which sites can be trusted with their sensitive information. Also, the repository facilitates the transfer of the user's information. The overall effect of the system is to inspire greater confidence in online participation among users who desire to protect their personal information.

ACKNOWLEDGMENTS

This thesis would not have been possible without the support from my wife Brandi who held down the fort during the many hours I spent at school. Thanks also goes to my daughter Amanda who always gave me a reason to smile after a long day in the lab. Sincere thanks goes to Dr. Kent Seamons for his deep insight and support as my advisor and employer. Lastly, I greatly appreciate the ideas and contributions offered by the fellows in the Internet Security Research Lab.

This research was supported by funding from the National Science Foundation under grant no. CCR-0325951 and prime cooperative agreement no. IIS-0331707, and The Regents of the University of California.

Table of Contents

List of Figures

Chapter 1 — Introduction

The remarkable growth in the number of online services over the last few years has encouraged more people to turn to the Internet for finding information and purchasing goods. As additional sites become available, people are more able to find services that satisfy their interests. Many of these services desire to create a relationship with their users, introducing the need to acquire personal information. At the same time, people are wary of giving out sensitive information in fear that it might be abused.

In the past, closed systems have been the predominant form of information control. A closed system is an environment in which proposed services are filtered by a gatekeeper before being offered to the members of the community. In this type of environment, the growth and viability of eligible services are controlled and maintained by a centralized authority. This allows one to predict the quality and trustworthiness of the service based on what is known about the gatekeeper's policies. However, this also allows for censorship and restrictions on the amount and types of services allowed. As an example, consider a highly specialized bookstore that seeks a strong reputation for its technical literature. This reputation helps customers predict the quality of any book found in the store. However, it is developed according to the subjective judgement of the owner of the bookstore by selling only the books he considers of worth, while refusing to sell those that are deemed less valuable.

The Internet, on the other hand, is an open system. There is no gatekeeper that filters content or services. Rather, anyone can publish anything or offer any service. Such an environment facilitates a proliferation of published opinions, ideas,

information, and services. However, because everyone (including malicious entities) has a voice, determining what and who is believable is a difficult challenge. Indeed, it can be difficult to distinguish sites that can be trusted to protect a person's information from those that cannot.

Identity theft from participating in online activities is a significant problem. Phishing attacks are increasingly more sophisticated. The incessant demand for new and fresh spam targets elevates the incentive for online businesses to sell personal information while turning a blind eye to the consequences.

As the power, flexibility, and availability of the Internet continue to expand, so does the threat of identity theft and other abuses of personal information. Thus, a user's willingness to participate in new and exciting opportunities on the Internet is dampened by the requirement to proceed cautiously, sometimes suspiciously.

1.1 Establishing Trust and Transferring Information

Two motivating examples demonstrate common viewpoints of the same problem faced by the average user while online.

Suppose a potential e-commerce customer, George, is shopping for a particular digital camera online. While browsing, George discovers a web site that appears to offer the camera for a better price than any other site. Despite the assurance from his web browser that his session with the remote server is secure, he may be somewhat suspicious that this site may not adequately protect his personal information. In this case, helping George gain confidence that the server will protect his personal information may be the single most important factor in facilitating the desired transaction between them.

Secondly, suppose Janice is an avid participant of a popular Internet auction site. She receives an email from the auction site indicating that they are completing a

review of all accounts. The email instructs her to click on the supplied link and login with her normal username and password to verify that her account is still active. She clicks the link, which takes her to the very familiar login page of the auction site. She is on her own to determine if the site she is looking at is the authentic login site of the auction or if it is the front of a sophisticated phishing attack.

In addition to the stress of determining the credibility of the remote server in each of these scenarios, a second problem exists. If a user decides to interact with the remote site, he is required to enter personal information. The repeated entry of this information can be monotonous and time consuming, which may further discourage would-be participants.

1.2 Introduction to TrustBroker

The solution to both of these problems can be accomplished using TrustBroker: a service that comprises 1) an online repository of user information, and 2) a protocol that serves to establish trust between remote entities and automatically transfer user information between them.

The online repository aids the user by establishing trust on his behalf. Once trust has been established between the web server and the repository, the repository discloses the user's requested information. If trust cannot be established, no information is revealed.

This process provides simultaneous solutions to the two problems previously defined. First, the user determines if he can trust the unfamiliar server by noticing if the web server is able to acquire any of his personal information. Second, the user's information is automatically sent to the remote server, eliminating the need for the user to provide it manually.

1.3 Thesis Overview

The remainder of this thesis is outlined as follows. Chapter 2 enumerates the related work. Chapter 3 explains the challenges and possible solutions associated with the design of each of the elements of a successful TrustBroker architecture. Chapter 4 delineates a proof-of-concept implementation along with a detailed description of what approaches were utilized and why. Chapter 5 comprises a thorough security analysis of the TrustBroker system, including its strengths and weaknesses. Chapters 6 and 7 include the conclusions of this thesis and possible avenues for future work.

Chapter 2 — Related Work

A significant amount of research has been devoted to the problem of information privacy and usability on the Internet. This chapter is devoted to introducing the current state of research in this field as well as what is currently available in the consumer market.

2.1 Transport Layer Security

Transport Layer Security (TLS) is a widely used protocol that performs two important tasks with regard to security and privacy. First, it offers a mechanism by which each participating entity can verify the identity of the other. This protects against malicious impersonation attempts. Second, it establishes a secure connection between the two parties over which sensitive information can safely flow.

These two tasks are critical to a secure online environment. However, the Public Key Infrastructure (PKI) upon which TLS depends is not perfect. There are numerous ways to circumvent the security features of a PKI [**?**]. In addition, while TLS can ensure that the line of communication is secure, it cannot guarantee that either entity will protect information divulged by the other.

2.2 Identity Management

Many ideas have surfaced to remedy the problem of managing user identity on the Internet in an attempt to make the Internet easier to use. One method is to use a local repository, such as a data and password manager. This repository stores a user's personal information and passwords in an encrypted file on the local computer and supplies the correct username, password, and other data to a given site as directed by the user. This enables the use of multiple strong (i.e., hard to remember) passwords because the user is not required to remember them. However,

a number of drawbacks have been discovered in such systems. First, the keys to all accounts and passwords are generally protected by a single password, so if the file can be downloaded and cracked off-line by a malicious person, unauthorized access to all accounts and information is granted. This is especially dangerous given the tendency for people to select weak passwords. Second, since that information resides on only one computer, a person will not be able to access their online accounts from any other computer. Third, the loss of this password file renders all accounts inaccessible.

Another method of managing a person's online identity is to federate a set of websites so that the username/password that grants access to any of them also grants access to the rest. Federated Identity Management is a broad field of research that incorporates a diverse set of motivations and objectives. The common theme, however, is to unify a group of distinct security domains so that the identity of a particular user may be recognized from one domain to another with little or no user intervention [**?**]. The research space is immense, due mostly to the huge diversity of needs and usage models that exist in large institutions today. One example is the centralization of directory and account services at the University of Pittsburgh [**?**]. Another example is federated access control for databases [**?**, **?**]. A third example is the desire for Internet-wide single sign-on access control for every Internet user [**?**]. Though the requirements of these systems are diverse, they share a fundamental ideology in that they mostly rely upon a centralized store of user information within a federated community.

Two significant players in this field are Microsoft's Passport [**?**] and the Liberty Alliance's Federated Identity System [**?**]. Despite strong similarities between their ultimate objectives, these two groups have approached the solution in remarkably

different ways. Together, they serve as contrasting case studies regarding the problems associated with federated identity systems and different ways to solve them. A short description of these two systems follows, including a brief analysis of their respective benefits and drawbacks.

2.2.1 Microsoft Passport

Microsoft Passport [**?**] is built around a central user information repository. Each person that wishes to participate must create a relationship (a user account) with this repository. Each business that wants to participate must establish a business relationship with Passport and implement the proprietary technology that interfaces with Passport's central user information store. When a user visits a site that is Passport enabled, he need only enter a username and password to authenticate. This username and password combination is universal to all systems that are Passport enabled. With this information, the site can retrieve other information about the person from the central Passport repository.

The key benefit of this solution is that the user has to remember only a single username and password. Also, the user's information is centrally located. Changing pieces of information is a simple process that can be done from any computer with an Internet connection and web browser.

Some potential drawbacks exist, however. First, each person who wishes to participate must rely on the privacy practices of Passport to protect his information from theft and wrongful distribution. There are no alternative repositories. Thus, if Passport proves less secure with user information than a person deems reasonable or necessary, that person only has the option to stop participating. Second, having all the user information in one place makes a very inviting target for attackers. Third, not every online company can feasibly participate in Passport because of

economic issues. According to Microsoft, participation in Passport requires a yearly $10,000(US) fee in addition to a periodic $1,500(US) compliance fee per URL that is Passport enabled [**?**]. This is in addition to any consulting/development costs needed to implement the technologies required by Passport. Clearly, this eliminates many small- to medium-sized businesses from participating. In the end, this prohibits the wide-scale adoption of this technology.

2.2.2 Liberty Alliance

The Liberty Alliance [**?**] was also formed to solve the problem of online identity management. Their primary goal is to enable a federated identity management system between business partners and consumers. This means that after authenticating to one site, a user may visit any of the site's federated partners without re-authenticating—his identity follows him to each site.

A problem with this solution is that a user's identity federation and single sign-on capabilities are constrained by whether the sites that he visits have chosen to participate in a federation. If they have not, or if they are not allowed to participate in a particular group, the user cannot enjoy these benefits. Additionally, a site may only belong to one federation. Thus, if one interesting site belongs to one federated group and another interesting site belongs to a different group, the user cannot enjoy the benefits of a federation that includes both sites.

The potential problems become apparent as this structure is logically applied to the real world. Consider the situation in which a site wishes to federate with more than one group. For example, an online clothing store may wish to federate with a bank that is in one federated group. It may also wish to federate with other online stores owned by the same clothing company. Though, the Liberty Alliance has mentioned the intent to resolve this, their specifications currently prohibit a site

from joining multiple federations.

Additionally, this model requires that the user already have an account with a site before he can participate in the federated system. In the common situation where a user is discovering new sites, such as with George in the introductory scenario, this forces the user to decide if he can trust the new server with his personal information before he can take advantage of any features of the federated system. Because of this, it may also discourage the active search for new Internet content and services.

Another issue arises from a business perspective. Since groups are formed among sites in order to establish single sign-on capabilities between them, the barriers to entry for a company may become very large. In some cases, groups could restrict entry to only those who meet very specific and perhaps strict criteria. For example, it may be possible that a group of large corporations decide that membership in their group requires a hefty fee. In this case, small- to medium-sized businesses are effectively banned from participating in that group.

Another example of failure might occur when a user desires to check airplane ticket prices from multiple competing websites. These companies, in competition with each other, may not desire to federate. Again, the system breaks and the user is not able to benefit from single sign-on technology.

In any of these scenarios, the core ideology of federated identity systems may become frustrated. Thus, for the average Internet user, the details of the Liberty Alliance specifications may restrict the desired user experience and undermine the real intent of federated identity systems.

2.3 Rating Systems

There has been a significant amount of research regarding the problem of establishing trust in an untrustworthy environment like the Internet [**?**, **?**, **?**, **?**]. Some

methods prescribe a rating system where users who view the content are allowed to rate it. Thus, the more a site is viewed and rated, the more accurate the rating will be with regard to the general consensus of the population. Amazon.com is one enterprise that uses such a rating system. However, since it is often a driven minority who are the most vocal, it becomes important to rate the raters [**?**]. Such a rating system becomes recursive as now you must rate the raters of the raters, and so on [**?**]. Some claim that this process should stop after the first recursion, with the raters of the raters; it becomes unmanageable to continue [**?**]. However, the level of reviewers at which the recursion stops cannot be held accountable for their actions because there is no further reviewing process.

An approach taken by Slashdot.org is to periodically select a groups of people at random from an established section of the general population to perform the ratings [**?**]. While this allows for a much more manageable rating system, it still may not convey the true value of the site with regard to the general population. Those selected to perform the ratings might be biased towards the feelings of those who chose them as raters.

Another fundamental hurdle with these types of systems is that immediate trust in new content or entities cannot be established. As mentioned, the rating of the site only becomes accurate over time as more and more users take time to view and rate the content. The Internet contains too much information to be rated piece by piece. The implementation of such a system on the Internet is infeasible.

2.4 Reputation Systems

A reputation system is loosely defined as a connected graph in which every node develops a reputation among its neighbors. While a fully connected graph would be ideal, the staggering number of nodes needed to represent the entire Internet

prohibits such a graph. Thus, each node in such a large graph is usually connected to a manageable number of neighbors.

Essentially, the reputation of a single node can be discovered by searching the graph until the node's neighbors are found and queried. When a particular node encounters an unknown entity, it may ask closely trusted neighbors if it should trust the new node [**?**]. The aggregated recommendation from its trusted neighbors becomes the new trust policy towards the new node, and a new connection in the graph is made. The proliferation of research in this field focuses primarily on the problems of identifying the attributes that can be used to compute trust and then transferring the meaning of that trust from one node to another.

Several approaches have been taken by researchers [**?**]. Among them, the common premise is that if a node can be trusted, all of the nodes that it recommends can be trusted, although possibly with a greater degree of caution. Further, all of the newly trusted nodes' recommendations should also be trusted. When trusted neighbor nodes disagree about whether an unfamiliar node can be trusted, an algorithm may be used to determine the likelihood that the new node can be trusted based on information from as many trusted nodes as possible. Thus, a reputation graph is established where any node can ask about the trustworthiness of any other node. The major benefit of this type of system is that it is scalable. As more nodes are added to the graph, the overall graph gets bigger, but only those nodes that are directly connected to the new node are affected by the increase.

Most reputation-based proposals acknowledge frustrating flaws. Primarily, the addition of a new node to the graph is a difficult task. It is difficult to create an initial trust classification that is accurate. In other words, it is difficult to bootstrap the system and difficult to add nodes in a reliable manner.

Another significant problem is that trust is context-sensitive. A person that is trusted to give advice about car repair may not necessarily be trusted to give sound financial advice, and vice versa. Both may be trusted, but only within their spheres of expertise. Thus, asking whether a node in the reputation graph can be trusted is meaningless unless a context for the trust is also provided. Thus, it is not enough to maintain a single trust rating for each neighbor. Each node must maintain a trust rating for each neighbor with regard to every context that may be of interest. This point necessarily breaks a reputation network into many layers, or many reputation networks, each layer representing a different context and intertwined with other layers. Therefore, a globally accepted list and definition of contexts must be maintained, as well as a mechanism for adding new contexts. Delimiting the entire list of possible contexts alone is likely an insurmountable task, not to mention the incredible task required of each node of maintaining its own location in many concurrent reputation systems.

Lastly, most algorithms for defining and transferring reputation-based trust are easy targets for deceptive nodes. In cases where some nodes may knowingly report false information, such as reporting that a node is trustworthy when it is not, or reporting that it is not trustworthy when it is, the reliability of information is severely damaging. Compounding the problem, there is usually no way to determine if a node is purposefully being deceptive, so the validity of the trust inference of nodes on the network may become highly questionable. This undermines the entire purpose of such systems [**?**].

2.5 Attribute-based Systems

Attribute-based systems for developing trust are relatively new. A fundamental observation that led to their inception is that identity-based systems are rigid, in-

flexible, and unscalable. When considering an open and dynamic environment such as the Internet, identity-based trust establishing systems are not a good fit. The reasoning follows: "Instead of trying to figure out *who* they are, let's care about *what* they are." Thus, if two entities are able to assert certain attributes about each other, a level of trust can be established even though their identities remain a mystery.

Trust negotiation [**?**] is an example of an attribute-based trust establishment model. The core process entails the iterative exchange of access control policies and credentials that attempt to satisfy those policies. These credentials usually take the form of certificates that are signed by a trusted third party and assert some attribute about the holder. Once the policies that govern access to a resource have been satisfied, the resource is granted to the requesting party. In this way, strangers are able to develop some level of mutual trust.

Attribute-based systems seem to play well in open and dynamic environments, and appear to be scalable and flexible. These attributes make them suitable for authentication and trust establishment models in open systems such as the Internet. However, there are some obstacles to overcome.

The policies that govern access to resources are generally very complex and difficult to create correctly. In addition to forming correct access control policies for sensitive data, credentials need to be obtained, managed, and protected. The loss, theft, or transfer of credentials is quite hazardous, and creates a quagmire for the management and distribution of credentials. In essence, these systems are hard to use.

Chapter 3 — System Requirements and Design

The exploration of the TrustBroker system design is divided into two sections: 1) the requirements analysis and 2) the design. The requirements analysis section provides a detailed exploration of what each component needs to accomplish to make TrustBroker successful. The design section analyzes alternative technological approaches that satisfy these requirements, and discusses the ramifications of design decisions on an actual implementation.

3.1 Requirements Analysis

This section discusses the overridding design goals and challenges of TrustBroker. It is important to first mention that while a privately managed, single-user repository is certainly a viable player in the TrustBroker system, this research focuses on the establishment of a public, multi-user repository. The arguments made hereafter are directed towards the multi-user paradigm.

User-friendly security is the fundamental principle of TrustBroker. While information security is essential, it must not compromise usability, and vice versa. The TrustBroker architecture has three basic components: repository, client-side module, and web server. Each of these is discussed in the following sections.

3.1.1 TrustBroker Repository

The TrustBroker repository must provide two remote interfaces. The first interface allows a user to register and manage personal information and privacy settings. In addition to these fundamental services, there may be ancillary services available to a user, such as options to view and manage a subscription or learn details about how the repository protects his personal information. Because usability is a fundamental design goal, this interface must be clear, simple to use, and comprehensible

to humans.

Remote web servers may connect in order to request information about a person through the second interface exposed by the repository. Through this interface the repository establishes trust with the remote server and handles incoming requests for personal information.

The repository must support find-grained access control for each piece of user information. It must guarantee that it will release information only to trustworthy servers.

3.1.2 Client-side Module

The client module resides on the client computer and provides a user interface to TrustBroker. The interface must be easy to use, yet leave the user in control over the disclosure of personal information. Thus, the design challenge is two-fold: 1) discover an appropriate way to grant maximum control to the user over the distribution of information; and 2) refrain from being intrusive or annoying when adhering to protective security practices. Additionally, the client module must operate correctly behind a firewall or from a private network.

3.1.3 Web Server

A web server interacts with the TrustBroker client module and repository by notifying the client module that it is TrustBroker-enabled and requesting personal information from a TrustBroker repository. When necessary, it must prove to the repository that it can be trusted to protect that information.

3.2 Design Analysis

The previous section outlined the TrustBroker requirements. This section presents design alternatives that satisfy the requirements, along with the pros and cons of each option.

3.2.1 TrustBroker Token

A significant design issue is to develop a strategy by which the repository can guarantee that the correct person's information is being distributed, and only to trustworthy servers. It must be able to guarantee that the client seeking services from the web server is who he says he is and that the server can be trusted to protect any disclosed information. To accomplish this, the repository must issue to the client some sort of immutable token, which can be passed to the web server instead of the requested personal information. The web server passes this token to the repository to indicate who's information is being requested.

Because this token is responsible to help guarantee the integrity of the system, the design and format of the token is critical. The token must provide these assurances:

1. The server must know which TrustBroker repository is representing the client.

2. The TrustBroker repository must be assured that the client making the request owns the token.

3. The token must provide proof of data integrity because it is transferred through a public, untrusted environment.

Any violations of these demands will negate the usefulness of TrustBroker.

The token contains 6 elements: the domain of the TrustBroker repository; the port on which the repository is listening; a user identifier; an expiration timestamp; a hash of the token (either keyed or signed); and an encrypted element that the client code adds (Figure 3.1). These elements are described below.

TrustBroker Domain. The first element of the token is the DNS domain name

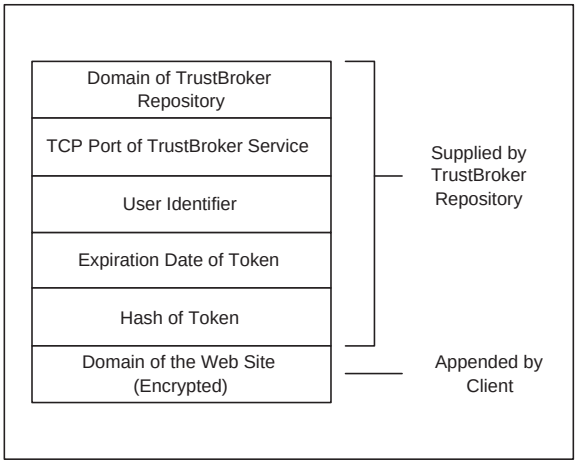

Figure 3.1: Elements of a TrustBroker Token.

of the TrustBroker repository. When the web server receives a token from the client, it uses the domain name to contact the appropriate repository.

TrustBroker Port. Instead of setting a fixed port for the TrustBroker repository, a repository may choose to listen on any port. This helps reduce the threat of a widespread Denial of Service attack against TrustBroker. Therefore, the second element of the token is the port on the TrustBroker domain to which the web server should connect when requesting data.

User Identifier. Some user identifier must be embedded in the token to identify the user to the TrustBroker repository. The following are options that satisfy this requirement:

Actual User Identifier. This user identifier could be the actual user identifier (such as the unique login name) of the user on the repository. However, this may provide in clear text unnecessary information to the web server and anyone else who gains access to the token.

Encrypted Form of Actual User Identifier. The next alternative is an

encrypted form of the client's user identifier as defined by the repository. This allows only the TrustBroker repository to determine the owner of the token.

Random Number The user identifier in the token could also be a random number associated with the user during the lifespan of the token. When the repository generates the token, it also generates a random number that is unique among all the users known to the repository. This number is temporarily associated with the client so that when the web server sends the token back to the repository, the TrustBroker repository can determine the owner of the token. This is a compelling option because it stores absolutely no information about the user, either in clear text or encrypted. Also, it removes the demand for an additional encryption/decryption from the TrustBroker repository. Lastly, this ID would change to a new random number with each token. This reduces the chance that it could be used to obtain unauthorized access to the client's sensitive information because every time a token is issued the identifier from the previous token is forgotten and therefore invalid.

The only overhead required would be to ensure the random number is unique to all other currently issued tokens.

Token Authenticator. A clear principle of TrustBroker's security model is to ensure that the contents of the token cannot be altered after it is created by the repository. Therefore, one element of the token is a value that can be used by the repository to determine if the token's contents have been modified.

One way to accomplish this is to sign a hash of the token's contents with the private key of the TrustBroker repository. This would allow any remote entity to verify the authenticity of the hash through decryption with the repository's

public key. Also, it would prevent anyone from spoofing the original hash by changing the contents of the token and recreating the hash, unless the private key of the repository was compromised.

The second form of protection is to use a hash-based message authentication code (HMAC). The repository maintains a secret key used when creating the token. When a request for information is received along with a token, the repository uses the secret key to verify the integrity of the token. This computation is very fast when compared to a database access or public key computation. This also means that no remote entity (e.g., the web server) would be able to independently check the integrity of the token.

More about the security aspect of this element of the token are discussed in Chapter 5.

Web Server Identifier. In order to generate the last element of the token, the repository generates a shared symmetric key and sends it to the user along with the token. The client uses this key to encrypt the web server's domain name using a strong symmetric-key cipher, such as the Advanced Encryption Standard (AES). This encrypted value is appended to the token. When the token is directed back to the repository from the web server, the repository attempts to decrypt this element using the key that is shared with the client. If the decrypted string does not match the domain of the web server as perceived by the repository, the token is deemed invalid.

This prevents anyone who may obtain access to the token from impersonating the client. Again, more about the security aspect of this element will be discussed in Chapter 5.

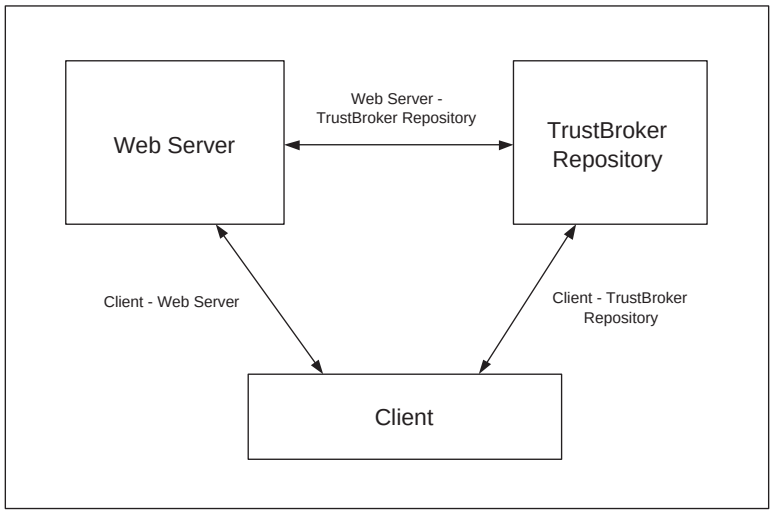

Figure 3.2: Channels of communication in a typical TrustBroker interaction.

Expiration Timestamp. When important digital documents travel encrypted or signed through public domains, they are often accompanied by an expiration timestamp. This is employed as an automatic invalidation mechanism so that the creator can ensure that the public is in possession of the latest version. Adding a timestamp to the token is a precautionary measure to ensure that the token cannot be stored for a long period of time while attacks against it or the repository are levied. It limits the window of possibility of a successful attack to the lifespan of the token.

3.2.2 Protocol

TrustBroker utilizes three communication channels (Figure 3.2). The first channel is for the user to request a token from the TrustBroker repository. Over the second channel the client sends his token and a request for a resource to the web server, and the web server responds to this request. The third channel occurs when the web server contacts the TrustBroker repository to request the user's information. To protect against eavesdroppers, all channels operate over secure communication

links. The following sections describe these three communication channels.

3.2.2.1 Client – TrustBroker Repository Communication

It has already been stated that a web interface needs to exist for the user to login and obtain a TrustBroker token. However, by forfeiting certain security and design goals, this does not need to be the case. Historically, the client computer has been quite vulnerable to attack, largely through the myriad of spyware that exists. To nullify this angle of attack on the TrustBroker system, a significant design goal is to remove all sensitive data from the client computer. However, if this were not important, or the threat could be dealt with another way, the username and password to the TrustBroker repository account could be stored on the client computer. The pair could either be stored in a password protected file or as browser settings. In this case, a mechanism could be built in which the client would simply click a button to retrieve a TrustBroker token. The user would not need to navigate to a login page or enter any information; a click of the button would suffice.

A slight twist to this model is to store the username and password on an external device, such as a flash drive. This way, the username and password do not reside on the computer but are available when the device is connected. This eliminates part of the risk because the data is not always available for exploitation. Instead the data on the device becomes a target only when it is connected to the computer, which might be infrequent. In this way, the benefits of single-click token acquisition might be realized with reduced security risk.

This research strives to minimize the risk of attack on the client computer. One way to satisfy this goal is to store all sensitive TrustBroker-related information in a secure online repository.

Assuming that a web interface is used to obtain a token from the TrustBroker

repository, the next design decision is to determine the mode of transport. One possibility would be for the client module to open a socket and attempt communication with the TrustBroker repository in a side band after the client has authenticated. The question then arises: who should attempt to make the connection? The client could try to connect to the TrustBroker repository or the repository could connect to the client. Allowing the repository to connect to the client may not be a good idea for several reasons. First, it opens an unnecessary socket on the client to the outside world, which is poor security practice and may make the client a viable target for attack. Secondly, there may be a firewall or private router that might block the connection attempt. Therefore, this does not seem like an appropriate approach. However, if the client attempted a second connection to the TrustBroker repository to request the token after the client had authenticated over the first connection, these problems are largely fixed. Indeed, this is a viable option, albeit somewhat complex.

The last possible option would be to transport the token within the same HTTP session as the login. The following is a list of mechanisms through which the token might be sent, along with an explanation of their strengths and drawbacks.

Browser Cookies. The token could be sent using the standard and well-used cookie protocol.

Advantage. The advantage of this approach is that browsers already support cookies. This reduces the amount of coding needed to create the client and web server modules.

Disadvantage. The cookie mechanism is designed to allow a web server to place a key-value pair on the client computer that will be made available only to the server that generated it when the browser returns. The transfer of

cookies is transparent to the user. Since the design goals of TrustBroker are to transfer tokens to sites that did not issue them and to grant the user control of the distribution of the token, the cookie mechanism seems ill-suited for the task of transporting the token.

Special header tags. A second way to transport the token is in special tags inside the HEAD portion of the document. The HEAD section is designed to include meta information about the page and is not displayed to the user. The token could be placed inside special TRUSTBROKER tags, or inside the standard META tags.

Advantage. The major advantage of this approach is its simplicity. Also, it requires the client-side module to scan only the HEAD section of a web page, thus reducing the time required to discover a token.

Disadvantage. No significant drawbacks to this approach are apparent.

3.2.2.2 Client - Web Server Communication

When sensitive information is needed to complete a transaction, the client sends the token instead. There are two main approaches to accomplishing this. Either the web server could actively collect the token from the browser, or the web server could passively notify the browser that it is enabled to receive TrustBroker tokens.

Allowing a web server to snatch a token without the user's knowledge violates one of the basic design principles, namely that the ultimate control of the token should be governed by the user. This would require the user to authorize or reject every token request from the web server. However, another design goal is to remain unobtrusive to the user. Thus, allowing the web server to actively collect the token does not permit the client module to simultaneously adhere to both of the fundamental goals

of TrustBroker: usability and security. One succeeds at the expense of the other.

The second approach is to create a notification mechanism that only informs the browser that the web server is capable of handling a TrustBroker token. The browser notifies the user in any number of ways, including methods that are unobtrusive and not restricted to yes/no dialog boxes. The token would be sent only when the user actively requests it. This approach seems to fulfill both of TrustBroker's fundamental design principles: the user retains full control over the distribution of the token and yet is not constantly bothered by obtrusive dialog boxes.

Given a passive notification method, the design of the notifier becomes important. The web server would need to embed the notifier into the returned web page and the client module would need to be actively looking for it. The following is a list of possible notification methods, with their advantages and drawbacks.

Hidden form field. One way to notify the browser that the server could accept a token would be to generate a form that includes a special hidden field. This hidden field would not have an initial value, but would have a standardized name that indicates to the client module that it should include the token as the value of that hidden field when the user clicks the submit button.

Advantage. One of the advantages of this approach is that it would require the client module to only scan web forms for the special hidden field. Since many pages do not have forms, the scanning would usually be very fast. Another nice aspect of this approach is that the client module would simply have to insert the value into the form. The basic functionality of web forms in browsers would do the rest and the token would be transmitted via the post data.

Disadvantage. The biggest drawback to this approach is that the user may

miss clues from the client module that his token is being requested. A malicious server could place a form with this special hidden field in every page it serves, trying to phish for any tokens it can acquire. The only way to stop this would be to generate increasingly obtrusive notification mechanisms. This interferes with the usability principle of the system.

HTML Elements. Like receiving the token, the notification mechanism could be passively embedded in the HEAD section of a web page, in either special TRUSTBROKER tags or the standard META tags.

Advantage. A strong advantage to this approach is that the client code could scan pages, notice when a web server is token-enabled, and notify the user in an unobtrusive fashion. No active response is necessary, and if the user ignores or misses a hint from the browser, there is no risk of accidentally sending his token. This allows the user to fully benefit from TrustBroker without being annoyed by obtrusive security measures.

Disadvantage. There are no apparent disadvantages to this mechanism.

3.2.2.3 Web Server - TrustBroker Repository Communication

The communication channel between the web server and the TrustBroker repository is the simplest because it involves no humans. Once this communication channel opens, the user has received a token and has willingly granted the web server authorization to use it in order to acquire his information. Even though the human usability of the system is largely irrelevant at this point because no human is involved in this transaction, there remain significant technical design challenges.

The main challenge is the communication protocol used between the web server and the repository. It is not limited to HTTP as other communication channels are;

any data transport protocol could be used. This raises the question of whether a proprietary protocol should be created specifically for this communication or if an existing, standardized protocol should be utilized. For the ease of implementation as well as for propriety's sake, the responsible answer seems to be that if a standardized communication protocol exists that could satisfy the demands of a TrustBroker interaction, it should be used.

While many approaches may suffice, messages formatted in XML and sent directly over the secure communication channel would provide a simple, flexible mechanism to transfer information.

Another decision regarding the communication protocol between the web server and the repository is the delineation of messages that are required to complete a transaction. There are three classes of required messages: an information request message, the required trust negotiation messages, and an information response message.

The final design element of this part of the protocol is the contextual language that is spoken. For example, when a web server requires the first name of the client, it must somehow encode this into the request so the repository can correctly extract the desired piece of data. Researchers have made strides in defining naming standards for commonly used pieces of information on the web [**?**]. One such set of standards is defined by the Electronic Commerce Markup Language [**?**], which standardizes the names of pieces of information that are commonly used in online commercial transactions. As the demand for transferring other kinds of information (such as medical and scholastic) increases, more standards will need to be defined and used. TrustBroker will need to adapt to these new standards as they are developed.

Additionally, there may be cases where a user desires to add non-standardized

fields to his repository of information, such as a username and password for a particular site. This would be highly beneficial against phishing attacks. An email that asks a user to login via a link in the email is suspicious in nature, and may be a sophisticated phishing attack. Under normal circumstances, a careful user would simply ignore the email or at least visit the site through normal methods (i.e., browser bookmark, typing the URL into the address bar, etc.). However, if the username and password were stored in the repository, the user could safely click the link from the web page. Instead of manually entering the login information, the user would initiate the TrustBroker system and let the remote site attempt to acquire the information from the repository. By allowing the repository to establish trust for him, a user could be protected against phishing attacks.

3.2.3 Data Obfuscation

Since a typical TrustBroker repository stores sensitive information about many people, it is a rich target for attack. If an attacker compromises the database and gains access to the information, significant damage could be done. One method of mitigating this risk is to obfuscate the data stored in the database so that a successful attack on the repository's database yields only encrypted, and therefore useless, data. Thus, the challenge is to grant only authorized people access to the data in a clear text form.

3.2.3.1 Protection from External Attacks

Protection against external attacks could be implemented by storing the data in an encrypted form. When a new user registers their data with the repository, the data is sent in clear text (but over a TLS connection to prevent eavesdropping). Once the repository has the data, it generates a strong key, encrypts the data in the database, and sends the key back to the client. The repository does not store this

key.

The key is appended to the token when it is sent to a web server. Thus, if the server is able to acquire information from the repository, it can decrypt it using the key provided by the client. If it cannot gain access to the data, it is left with the key, but without access to the encrypted information.

With this system, the data cannot be acquired directly from the repository's database in clear text form. The key from the client is also necessary. This raises the required level of attack sophistication, as now an attacker must obtain unauthorized access to the information and acquire the appropriate user's key.

In addition, if an attacker (namely the web server, as it is often granted access to the token) was able to acquire the key from a token, and gain unauthorized access to the repository's database, it could still obtain information for only a single user. The information of other users would remain secure because it is encrypted with unique keys.

Of course, once the web server has a user's key, it is no longer secret. To mitigate the risk caused by repeatedly distributing the key, the user would periodically renew the encrypted data with a new key.

The downside of this approach is that the loss of the key on the client side means the data cannot be retrieved. This would require the client to re-register his information with the repository.

Despite the fact that the key is being released to some web servers, it is still prudent to minimize its distribution. A second downside to this approach is that malicious web servers that are able to acquire the key may readily distribute it to other servers. Also, the fact that the client stores the key locally means that it is now an avenue for attack.

While there are drawbacks to this approach, its goal is to protect personal information from direct attacks on the database and attacks on the repository service. To protect only against attacks on the database, any number of commercially encrypted databases could be employed.

3.2.3.2 Protection from the Repository

Another way to encrypt the user's data in the repository is for the client to generate its own key, and send only encrypted data to the repository when the client first registers. Thus, the repository never sees the data in clear text. The key would be sent in the token so the web server could decrypt the information, as in the previous example.

In addition to the problems of the previous example, the drawback to this solution is that it begins to encroach upon the basic principle of usability. The user would need to be more heavily involved with the generation of the key and the encryption and transmission of the data.

3.2.3.3 Protecting Against External and Internal Threats

To protect against both external and internal threats, this idea of data obfuscation could be carried one step further. As in the previous section, the data would be sent to the repository encrypted with a key generated and stored by the client. However, in this case, the client would not append this key to the token. Instead, when the server obtains the information from the repository, it would send the data back to the client for decryption. The client would decrypt the information with its secret key and return the data to the web server in clear text.

This solution solves the key distribution problem as noted in previous solutions. However, the complexity of the communication is significantly increased.

A further study of the benefits and risks of attempting such modifications to the

proposed TrustBroker system is left to future work.

3.2.4 Repository Policy Agglomeration

Each TrustBroker repository may allow separate pieces of a user's information to be protected by individual policies. For example, it is likely that a person would want the repository to protect his credit card information more strictly than his first name. In this case, the policy protecting the release of his credit card information would require a higher level of trust than the policy governing the access to his first name. Thus, separate policies might be applied to disparate pieces of information. Therein lies a challenge for the TrustBroker repository to assimilate the information being requested and unlock only the pieces that the web server is authorized to see.

Several options are available. First, the repository may send all of the policies that govern all of the requested data to the web server. The web server would look through the policies to determine which ones it thinks it can satisfy.

This approach reduces the number of iterations needed to establish trust. However, it also requires the web server to employ more resources during this portion of the transaction because it must logically analyze the set of policies received from the repository.

Another approach is to employ a linear method, wherein the policy that governs each piece of data is sent to the web server in turn. This allows the policy analysis to remain on the repository, but may cause significant delays in the transaction due to the number of rounds that may be required.

In some instances, it may make sense to assume that policies can be ordered. In other words, the statement "the credential that satisfies policy A offers greater privacy assurance than the credential that satisfies policy B" may be possible for every policy available. When this is the case, several approaches seem reasonable.

The first would be to challenge the web server with the protecting policies in descending order starting from the most stringent. If the web server could satisfy the most stringent policy, it could be inferred that it could satisfy the less stringent ones. Therefore, the trust negotiation session could stop and the web server would be authorized to see all the data requested. If it cannot satisfy that policy, the next easiest policy would be attempted. This would continue until the web server is able to satisfy the policy, at which time the trust negotiation would stop and the corresponding data would be delivered. The possible benefit is that only one round of trust negotiation would be necessary for servers that were able to satisfy the most stringent policy. However, the drawback would be that the trust negotiation session with web servers that possess no satisfactory credentials might take a long time.

Thus, a second method would be the exact opposite: start the trust negotiation with the least strict policy, then iterate through consecutively stricter policies until the web server fails to satisfy it. The repository could safely assume that the web server wouldn't be able to satisfy any others and could terminate the session and return the data tied to the policies that had been satisfied. The potential benefits and drawbacks are the opposite of the previous method. Detecting web servers that are not very trustworthy might occur very quickly. However, the trust negotiation session with web servers that are highly qualified to protect user information may take a long time.

A happy medium might be to begin in the middle of the policies, with regard to their strictness. If the web server could satisfy that policy, stricter policies could be challenged. If not, less strict policies could be challenged until the web server finally satisfied one. This way, the worst-case time it would take to determine the level of privacy assurance that a web server could guarantee would be half the worst-case

time of either of the previous two alternatives.

A slight improvement might be made by performing the equivalent of a binary search through the policies. This would allow the repository to establish trust in logarithmic time.

Chapter 4 — Implementation

In order to establish the practicality of the TrustBroker system, a proof-of-concept implementation was developed. This chapter presents the implementation's details, especially as they relate to the design principles mentioned in the previous chapter. The explanation of the implementation is divided into the three core pieces of a TrustBroker system: the client-side module, the TrustBroker repository, and the web server.

Although several trust establishment protocols are feasible for the web server and TrustBroker repository to establish trust, the implementation uses attribute-based trust negotiation as its underlying trust establishment mechanism.

While the TrustBroker design calls for TLS sessions between each entity, they are not necessary to establish the feasibility of a commercial TrustBroker system. Therefore, they were not included in the implementation.

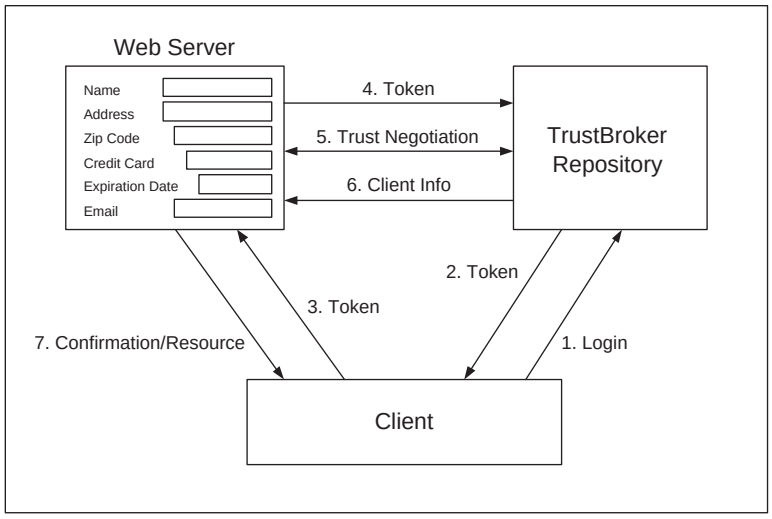

Figure 4.1: Overview of a TrustBroker transaction.

The implementation supports the TrustBroker system architecture shown in Figure 4.1. First, the client logs in to the TrustBroker repository (1) and is issued a token (2). When the client requests a resource from a web server, the token is also sent (3). The web server forwards the token to the TrustBroker repository and requests the necessary information to complete the transaction (4). TrustBroker verifies whether the web server satisfies the policies governing the client's information (5). If the web server satisfies the required policies, it is granted access to the requested data (6). The server asks the client for confirmation of the data and grants access to the resource (7).

4.1 Client-side Module

The client module is an extension to the Mozilla Firefox web browser and provides two key functions. First, the user interacts with the system via a graphical interface that consists of a browser toolbar and a submenu added to the default browser tool menu. Second, the client module manages the acquisition, storage, and transmission of a token.

4.1.1 User Interface

All of the client module functionality can be accessed via the tool menu (Figure 4.2). It includes the ability to request, send, delete, and view the contents of the token; adjust user settings and preferences; and see details about the installed TrustBroker client module.

The first two items allow a user to request and send a token, respectively. The third item on the menu ("Delete Token") allows the user to delete the current token. This option is valuable in situations where the client may want to explicitely remove the token from memory, such as when using a public computer. The "View Token" item allows the user to view the contents of the token, which may help the user

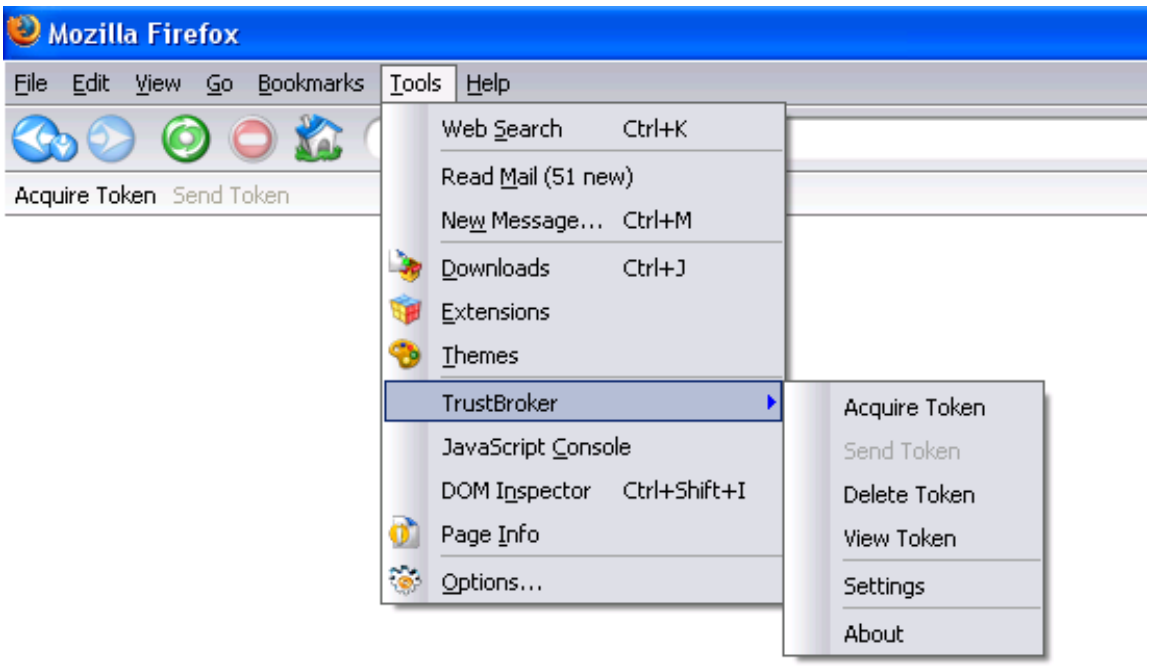

Figure 4.2: TrustBroker tool menu.

Figure 4.3: Client module About dialog box.

(a) "Send Token" button disabled.

(b) "Send Token" button enabled.

Figure 4.4: TrustBroker toolbar.

determine if his current token has expired, but is provided mostly for development and debugging purposes. The fifth item allows the user to adjust user settings and preferences, which will be discussed shortly. The last item causes an informational dialog box to be displayed that has detailed information about the client module, such as the author, its version number, and where to go for updates (Figure 4.3).

The second main graphical element of the client-side module is an optional toolbar, which duplicates functionality found in the tool menu (Figure 4.4). Its purpose is to provide quick access to the two most common user actions: acquiring the token and sending it.

The "Acquire Token" button is used to request a token from the TrustBroker repository. When clicked, the browser navigates to the login page of the user's repository. If the user already has a valid token, a dialog box will appear asking the user to confirm that he would like to revoke the current token and acquire a new one. This dialog box is necessary because the circumstances in which a new token is desirable over the current valid one are rare and it is likely that the user clicked the button by mistake or as a mechanism to check if he currently has a valid token.

There is another aspect of the "Acquire Token" option that increases the system

usability. Before the browser navigates to the login page of the user's TrustBroker repository, it stores the current URL. After acquiring the token, the client module navigates the browser back to this URL. This allows the user to navigate to any page before acquiring a token, and he will be returned to that page directly after receiving it.

Since most sites do not require personal information, sending a TrustBroker token to each server is undesirable. Because of this, the "Send Token" option is disabled by default (Figure 4.4(a)). However, when a site is TrustBroker-enabled (a description of how a site becomes TrustBroker-enabled is found later in this chapter), the "Send Token" button is enabled (Figure 4.4(b)). This allows the user to send his token instead of manually submitting personal information. Thus, when the user expects a site to be TrustBroker enabled, such as when filling out a web form, he may verify it by checking the status of the button. If enabled, and if he desires, he may click the "Send Token" button to send the token instead of entering data in the form.

This method of transferring the token is both secure and easy. It grants the user strict control over when the token is sent, while remaining unobtrusive during normal Internet use.

The fifth item on the menu opens a dialog box that allows the user to view and modify settings that determine the behavior of the client module (Figure 4.5). These settings are stored as part of Firefox's browser settings and are loaded and stored like other user preferences. Figure 4.6 shows the TrustBroker-related settings displayed alongside other user-configurable settings.

The *trustbroker.enabled* property allows the user to enable/disable the client module. When disabled, the client module becomes dormant, hides the toolbar, and does not respond to any TrustBroker-related event.

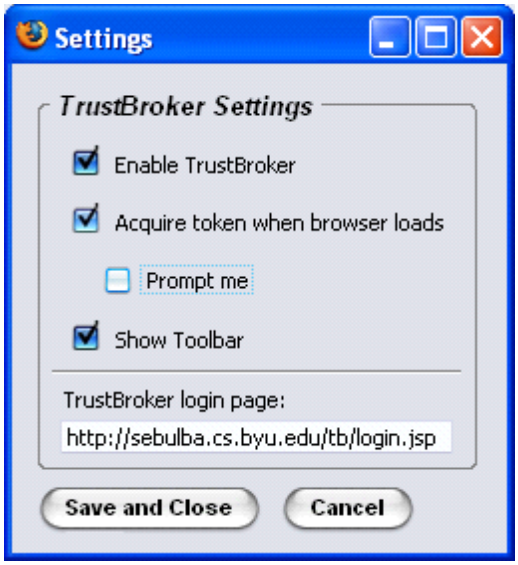

Figure 4.5: Client module Settings dialog box.

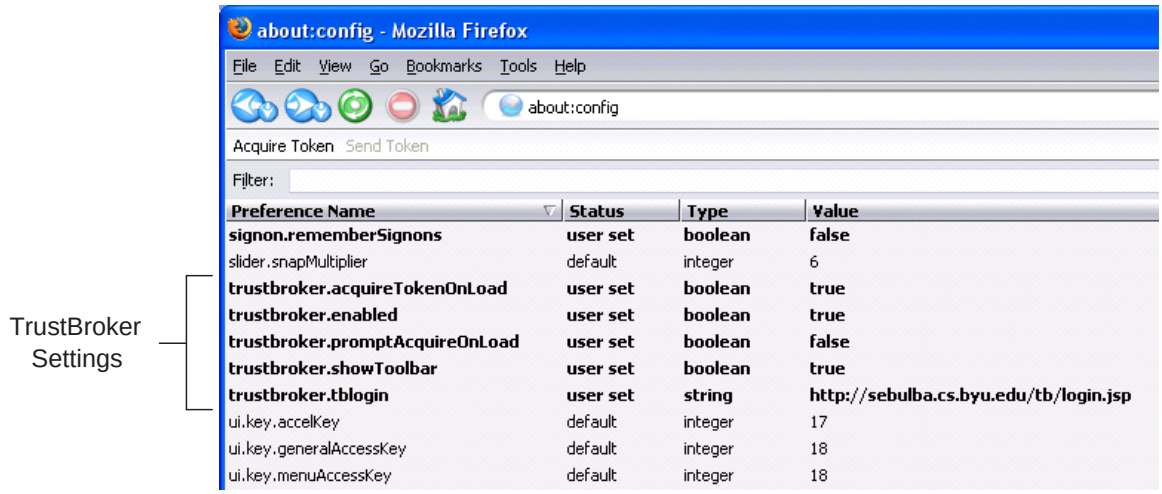

Figure 4.6: TrustBroker settings as part of web browser preferences.

Figure 4.7: Confirmation dialog box when the browser initially loads.

If the *trustbroker.acquireTokenOnLoad* property is set to true, the browser will immediately navigate to the TrustBroker repository's login page when it loads.

However, if the *trustbroker.promptAcquireOnLoad* property is true, the user will be asked to confirm his desire to obtain the token when the browser loads (Figure 4.7). At this point, the user will be able to cancel the automatic request for a token if he desires.

The *trustbroker.showToolbar* property allows the user to determine if the toolbar should be visible. If the toolbar is hidden, the user can still interface with the TrustBroker system as before through the tool menu.

The *trustbroker.tblogin* property stores the URL of the TrustBroker repository's login page. When the user clicks the "Acquire Token" button on the toolbar or tool menu item, the client module directs the browser to the specified URL. If no URL is specified, a dialog box will request that the user enter a URL.

Again, these settings are modified by the user from the settings dialog (Figure 4.5).

These graphical elements of the client module combine to create the user interface to TrustBroker. It allows the user to remain in control of the token, yet is unobtrusive in that the user is not bothered by repeated pop-up boxes or security

```
<meta trustbrokertoken="[token]::[shared_key]"/>
```

Figure 4.8: Sending a token to the client.

notifications.

4.1.2 Token Management

The second aspect of the client-side module consistutes its interface with the TrustBroker system. This entails being able to request, interpret, and send the token when needed. Each of these tasks will be explained in detail.

4.1.2.1 Requesting and Interpreting a Token

A token request is always initiated by the user clicking the "Acquire Token" button on the toolbar (Figure 4.4) or by selecting the menu item from the tool menu (Figure 4.2). The only exception to this is when the user specifically asks the browser to request a token when it is first loaded (Figure 4.5). In any case, when this request is made the client-side module temporarily stores the current URL, including the query string if it exists, and navigates the browser to the TrustBroker login page as specified by the URL in the settings panel (Figure 4.5).

From that point on, the client-side module scans the HEAD element of every page visited, looking for a token sent from the repository. After discovering the token, the module redirects the browser to the stored URL

The repository sends a token to the client by including a special META tag in the HEAD element of any web page (Figure 4.8). The token consists of a string concatenation of the five required elements listed in Section 3.2.2 delimited with the '@' character. The secret key is the 128-bit key shared between the repository and the client, which is used to generate the sixth element of the token.

When receiving this token, the module checks to make sure the domain sending

```
<meta token_destination_url="[URL]"/>
```

Figure 4.9: Notifying the client that a site is token-enabled.

the token is the same as the domain of the user's repository as specified in the settings panel. This prohibits any arbitrary website from replacing the token.

4.1.2.2 Sending the Token

The client module allows the token to be sent to any website that is token-enabled. In order to be token-enabled, a website must include a special META tag in the HEAD element of the web page (Figure 4.9). When the module detects this special tag, it enables the "Send Token" button on the toolbar (Figure 4.4) as well as the "Send Token" option in the tool menu (Figure 4.2). If a user desires to send the token, he simply clicks either of these two controls. The module encrypts the domain of the website, appends this encrypted value to the token, and sends the token to the site indicated in the value portion of the META tag attribute as a query string. After sending the token, the module waits for a response from the web server. The response consists of a URL that points to the location where the client may continue the transaction process or be notified of any problems that may have occurred.

4.2 TrustBroker Repository

There are two pieces of the TrustBroker repository's implementation. First, people use a web interface to register their information and request tokens. Second, web servers request/receive client information and perform trust negotiation through a remote server interface.

4.2.1 Web Interface for Users

This portion of the implementation is straightforward. It consists of a few web pages specially created to support the TrustBroker system. The first is a login page. This page has a simple username and password form that allows a person to login and request a TrustBroker token. Also on the page is a link to the registration page where a user may register his information with the TrustBroker repository.

The login attempt is sent through form data to the second web page, the login confirmation page. This page authenticates the username and password. If they are valid, it also generates a token and shared key unique to that user and token. The token and key are placed inside the special META tag in the HEAD element of the page that is sent back to the client (Figure 4.8).

When the user desires to register, a form is provided so the user can enter all the personal information he wants to store and allows him to assign each piece of data a protection level. Each protection level is translated into a policy that governs access to the data. Thus, the higher the protection that a user demands on a piece of information, the stricter the policy is. This mechanism allows a user to determine how strictly each piece of information is protected. The current implementation stores the user's information in clear text in the database.

4.2.2 Remote Interface for Web Servers

The TrustBroker repository's second interface consists of a simple TCP socket. This socket listens on an arbitrary port, chosen by the repository. This port is included in the token, to ensure the web server is aware of how to establish the connection.

Through this interface, a series of messages are exchanged that lead up to and include the tranfer of user information. The messages are formatted in XML for

```
<?xml version="1.0" encoding="UTF-8"?>
<trustBroker type="clientHelloMessage">
<token>
TrustBrokerDomain@TrustBrokerPort@UserId@ExpirationDate@HMACOfToken@EncryptedWebServerDomain
</token>
<ECMLData type="Ecom_ShipTo_Postal_Name_First"></ECMLData>
.
.
.
<hellomessage ...>
...  standard trust negotiation client hello message ...
</hellomessage>
</trustBroker>
```

(a) Initial Request for Information / Trust Negotiation Client Hello Message

```
<?xml version="1.0" encoding="UTF-8"?>
<trustBroker type="serverHelloMessage">
<hellomessage ...>
...  standard trust negotiation server hello message ...
</hellomessage>
</trustBroker>
```

(b) Initial Request for Information / Trust Negotiation Server Hello Message

```
<?xml version="1.0" encoding="UTF-8"?>
<trustBroker type="tnTermination">
<ECMLData type="Ecom_ShipTo_Postal_Name_First">Michael</ECMLData>
<...  standard trust negotiation termination message ...
</trustBroker>
```

(c) Transfer of User Information / Trust Negotiation Termination Message

Figure 4.10: Protocol messages between web server and TrustBroker repository

easy parsing. There are four types of messages that are used to carry out this communication:

Initial Request for Information / Client Trust Negotiation Hello. After receiving a token from the client, the web server will send this message to the repository to request the client's information. The contents of this message include the token, a list of desired data items in Electronic Commerce Markup Language (ECML) notation, and the contents of a standard Trust Negotiation Client Hello message (Figure 4.10(a)).

Server Trust Negotiation Hello. Upon receipt of the client hello message, the repository evaluates the validity of the token according to the following steps.

1. It computes the HMAC of the received token using the secret key created when the user obtained the token and compares it to the original HMAC generated at the same time. Since the secret key is required to produce a correct HMAC, a matching comparison guarantees the token's authenticity (as long as the repository has safely protected the key). This check protects against modification attacks.
2. It decrypts the signed domain name with the shared key associated with the user indicated in the token and compares that name with the domain name of the server that is requesting the information. As long as DNS remains reliable, this domain name cannot be spoofed because TLS guarantees domain name authenticity. This comparison protects against impersonation attacks.

After establishing the validity of the token, the repository evaluates the requested data items and determines all of the policies that govern their dis-

closure. In the event that the information is guarded by separate, but hierarchical policies (meaning they can be ordered in the level of protection they offer), the server determines which policy to try first and integrates that into the standard Trust Negotiation Server Hello message (4.10(b)), which is then sent to the web server.

If the token is not valid, the server sends the Trust Negotiation Termination message indicating that it is not prepared to offer any information to the web server.

Trust Negotiation Continuation. After the hello messages are exchanged, trust negotiation ensues. The repository determines how much to trust the web server according to the policies the server satisfies during trust negotiation. Trust negotiation is completed when the repository can satisfactorily determine how much to trust the web server.

Trust Negotiation Termination / Transfer of User Information. Trust negotiation can be terminated at any time by either side. The repository terminates the negotiation when it discovers the level of trust that may be applied to the web server. The web server may terminate the negotiation if it becomes too lengthy, resource-intensive, or for any other reason. When either side decides to terminate trust negotiation, it sends this message to the other party.

The Trust Negotiation Termination message contains the standard trust negotiation termination information. If the TrustBroker repository is terminating the communication, the requested data that has been unlocked via trust negotiation (Figure 4.10(c)) is also included. Requested data fields that are

protected by policies that were not satisfied by the web server are included but left blank, indicating that the repository either does not have the requested data, or refuses to disclose it. The web server may notify the client if any necessary information could not be acquired or if any errors occurred.

4.3 Web Server

There are three basic requirements that a web server needs to satisfy in order to interface with the TrustBroker system. First, it must notify the client module that a page is TrustBroker-enabled. Second, it must scan incoming resource requests for TrustBroker tokens. Third, it must be able to negotiate trust with a web server to acquire a user's information.

4.3.1 Page-enabling

As mentioned previously, the mechanism for enabling a web page is to include a special META tag in the HEAD element of the page (Figure 4.9). This tag indicates to the client module that the current site is capable of handling a TrustBroker token instead of any requested information, and provides the URL where the token should be sent.

4.3.2 Handling the Token

When processing a request, the token-enabled page checks to see which kind of data has been supplied. If standard form data has been sent, the transaction proceeds as normal. If the request includes a TrustBroker token instead, the server uses this token to request the desired information from the TrustBroker repository.

4.3.3 Acquiring the User's Information from the Repository

If a resource request includes a TrustBroker token in place of the expected form data, the web server must use the token to acquire the user's information from the TrustBroker repository. To do this, it must make the appropriate request and

establish itself to the repository as a credible server. Trust negotiation is used as the method of establishing this credibility.

The current implementation of TrustBuilder, a package developed by the Internet Security Research lab at Brigham Young University to perform core trust negotiation tasks, is written in the Java language. To leverage the benefits of a homogenous solution, the web pages that handle TrustBroker tokens are also written in Java, in the form of JavaServer Pages (JSP). Thus, the Java packages that are required to support trust negotiation (including TrustBuilder) are loaded directly into the environment where the JSPs are compiled and executed (Apache's Tomcat, in this case).

When a token-enabled web page discovers a token in the request, it parses the token and makes a TCP connection to the specified TrustBroker repository on the specified port. From there, it sends the *Initial Request for Information / Client Trust Negotiation Hello* message (Figure 4.10(a)) to the repository, thus initiating the communication between the web server and the repository.

The repository challenges the credibility of the server by asking it to satisfy the security policies governing the requested data. If the repository determines that it can trust the web server, it releases the desired information.

Once the *Trust Negotiation Termination / Transfer of User Information* message (Figure 4.10(c)) has been received, the JSP closes the connection to the repository and analyzes the client's information it received. At that point, it may ask the client to confirm the data or provide more information.

Chapter 5 — Threat Analysis

In the search for usability, the security of a system must not be compromised in any way. Thus, a thorough security analysis of TrustBroker is imperative. In most cases, a security model cannot formally be proven correct. Instead, it is analyzed with respect to the known attacks and vulnerabilities of other systems as applied to the new system. The longer it withstands this scrutiny, the more secure it is considered.

This chapter presents a threat analysis of TrustBroker. The client, web server, and TrustBroker repository are each analyzed for possible infractions in correct security principles. There are numerous well-known security threats that are not addressed by TrustBroker. Although some of these attacks are analyzed, the focus of this chapter is on threats introduced by TrustBroker.

5.1 TrustBroker Token

The token issued by the TrustBroker repository is a target for attack. It is a key element to TrustBroker's security, and is the focus of the threat analysis. There are three classes of attack that could be levied against the token: impersonation, replay, and modification.

5.1.1 Impersonation

An impersonation attack occurs when an attacker convincingly assumes the identity or characteristics of another person.

Consider the steps of the impersonation attack illustrated in Figure 5.1. The client requests a resource from a malicious web server posing as a legitimate service provider, and sends his TrustBroker token instead of personal data (A). The normal behavior of a legitimate web server would be to send this token to the TrustBroker

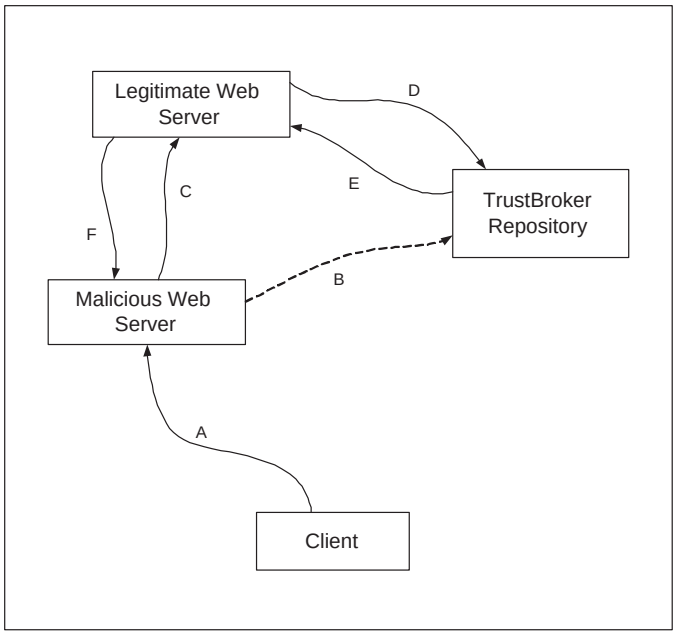

Figure 5.1: A basic impersonation attack using a TrustBroker token.

repository and request the client's information (B). Instead, the malicious web server requests a resource from a legitimate web server and attempts to impersonate the client by sending the client's token (C). The legitimate web server sends the token to the designated TrustBroker repository and requests the client's information (D). The TrustBroker repository authenticates the legitimate web server and releases the information (E). The malicious web server gains access to the client's information and is granted services from the legitimate web server using the client's identity (F).

However, the TrustBroker system design prevents this type of attack. Consider the sixth element of a TrustBroker token (Figure 5.2). It consists of the domain of the intended token recipient encrypted with a secret key shared only between the TrustBroker repository and the client. When a TrustBroker repository receives a token and request for information it decrypts this element and compares it to the domain of the web server making the request. If they do not match, the request is

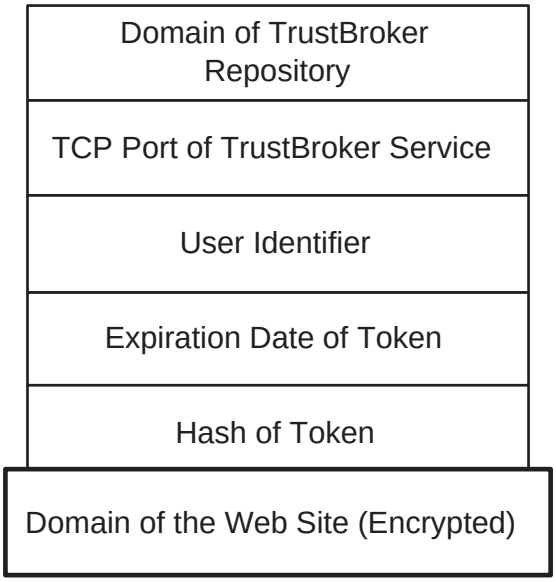

Figure 5.2: The sixth element of the TrustBroker token is the domain of the intended token recipient encrypted with the secret key shared between the client and TrustBroker repository.

denied. A third party attempting to impersonate a client will not be able to generate the correct value because it does not have access to the shared secret key. As long as the secret key is protected, only the client will be able to correctly generate this element. Therefore, impersonation attacks are not possible.

5.1.2 Replay

A replay attack is successful if the token is captured by a malicious entity and successfully re-used without the consent of the client. Since all TrustBroker-related communication occurs over a secure channel, the only substantial threat is a malicious, but trusted, web server.

If the web server is malicious, but cannot satisfy the policies governing access to the client's data, the token is of little value anyway and no replay attack is possible. If the web server is able to satisfy the policies that govern access to the client's personal information, it is unlikely to be malicious because the credentials necessary for unlocking the user's information would not be granted to a server with a history of malicious behavior.

5.1.3 Modification

A modification attack is successful if a malicious entity is able to modify the contents of a TrustBroker token such that a different person's information is accessed. This attack could be attempted by either a malicious client or web server.

The fifth field (Figure 5.3) of the TrustBroker token prevents such an attack. This authenticator field contains either a digital signature or an HMAC (i.e., keyed hash) that only the repository is able to generate. The repository can detect when a token has been fabricated or modified by recreating this authenticator from the contents of the token and checking its validity.

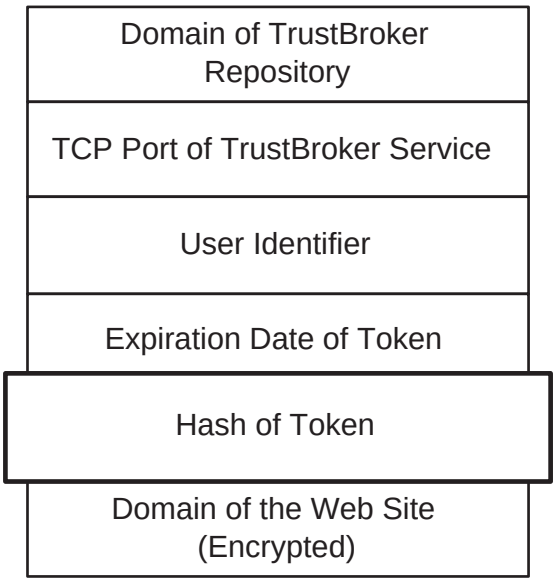

Figure 5.3: The fifth element of the TrustBroker token is a hash of the token's contents.

5.2 Pre-existing Threats

There are numerous well-known attacks that can be levied against many systems, including TrustBroker. This section briefly discusses some of them that are most relevant to TrustBroker.

Defective software. Experience shows that the design of a secure system can be rendered ineffective through faulty implementation. Faulty software on any of the entities involved could be exploited to break the TrustBroker system, either by gaining access to a user's information or by obtaining and abusing an otherwise valid token. This is especially dangerous on the client computer and the repository. The greatest protection against such problems is to use reputable software and keep up-to-date on software patches. Still, there is no way to guarantee that the software is free from error.

Spyware and other malicious software. Malicious software running on any of the three entities could be dangerous. The client could give up the token, secret shared key, username and password of the user with a simple keystroke logger or Trojan horse. The web server and repository could both disclose the sensitive information of the client. However, the implementation of standard security practices (such as firewall, antivirus, and intrusion detection solutions) minimizes this threat.

Malicious Insider. The problem of trusted, yet malicious, employees is well-known and difficult to solve. Often the measures taken to solve this problem collide head-on with the privacy and productivity of the employee. A malicious employee on the web server or the repository could gain access to user information with the intention of abusing it. With regard to TrustBroker, a possible solution to the malicious insider is to obfuscate the data on the repository, as discussed in Section 3.2.3. Such an effort would help ensure that the repository does not provide information to attackers, either external or internal.

Public Key Infrastructure. Numerous vulnerabilities exist in any Public Key Infrastructure (PKI) [**?**]. To the extent that TrustBroker relies on a PKI, problems such as private key compromise and faulty certificate generation might reduce the security of TrustBroker. This could become a problem with regard to the TLS sessions that exist to protect the transfer of information from eavesdroppers. It could also become a problem when an attribute-based form of trust establishment (such as trust negotiation) is used to build trust between entities. In either case, a server might be able to impersonate the identity and/or attributes of another server if the PKI model is compromised.

Distributed Denial of Service. Distributed Denial of Service (DDoS) attacks are among the most common form of successful attack against online entities. They certainly are a viable form of attack in the TrustBroker system. They could target a particular website (as the majority do now) but a potentially more debilitating attack would be against a specific repository. If the repository became inoperable, the advantages of TrustBroker would become moot. The success of website-based DDoS attacks is largely due to the fact that web servers listen on a standard port (port 80). Since the TrustBroker system allows the repository to select any port to listen on, this threat is greatly reduced.

Chapter 6 — Conclusion

As the Internet continues to grow in the number of available services, more people are participating in online activities. Unfortunately, the Internet can be a hostile environment, leaving the average user without help in protecting sensitive information. In many cases, the choice is to risk the possible abuse of this information or simply not participate. Similarly, phishing attacks are becoming more sophisticated, sometimes fooling even savvy users.

Additionally, the transfer of information is often a tedious task. Unlike physical stores where a user is required to divulge minimal information when making a purchase, online stores often require the user to fill out extensive forms that contain information about identity, physical location, and preferences.

TrustBroker is a system that seeks to alleviate these problems simultaneously. It is a system that utilizes a trust establishment mechanism to help the user gain confidence in the trustworthiness of a remote server, and automatically transfers the user's required information once trust has been established.

TrustBroker allows a user to obtain a token from a trusted online repository that stores his personal information. When online services request his personal information to complete a desired transaction, he submits his token instead. The server attempts to exchange this token for his information from the online repository. Before releasing the information to the server, the repository will attempt to establish trust that the server will not abuse this information. Once this trust has been established, the user's information is released to the server.

The average Internet user should have greater confidence while participating in online transactions involving TrustBroker. A user stores his sensitive information

in a TrustBroker repository, allowing him to benefit from the trust establishment mechanism without being required to understand its details. Also, he is generally not required to submit any information to the server, as it can acquire the information it needs from the repository if it demonstrates that it is trustworthy.

This research attempts to simultaneously support the seemingly contradicting design goals of security and usability. It presents design alternatives of a successful TrustBroker system, and analyzes the pros and cons associated with each approach.

The implementation presented in this research demonstrates the practicality of TrustBroker. It uses trust negotiation as the mechanism to establish trust, and a browser extension that allows the user to interface with the system. TrustBroker remains unobtrusive while protecting the information of the user and allow the user to remain in control of each transaction.

TrustBroker bolsters the necessary confidence to participate in new and exciting online activities by establishing trust on their behalf, and automatically transferring the necessary information when required. Additionally, this is all done with minimal user intervention, which allows the average user to participate without being required to learn the details of difficult and complex trust establishment mechanisms.

Chapter 7 — Future Work

TrustBroker establishes a framework for extensive future research. Much of this research has been mentioned in this paper. However, there remains significant areas of research that would serve to expand and strengthen TrustBroker.

The current design and implementation of TrustBroker utilizes trust negotiation as the trust establishment mechanism because it works well in open systems, such as the Internet. However, the trust establishment mechanism is independent of the TrustBroker framework, and may be interchanged with other trust establishment mechanisms without affecting the rest of the system. An interesting exploration would be to consider the benefits and drawbacks of employing other trust establishment mechanisms. Rating and reputation systems are promising prospects.

Two other research topics that are currently being pursued in the Internet Security Research Lab at Brigham Young University could be useful extensions to TrustBroker: digital receipts and context-sensitive trust negotiation.

Digital receipts are the digital equivalent of a receipt obtained as a record of transaction from a physical store and serve as proof of a transaction conducted online. Considering that web servers contact TrustBroker repositories directly when requesting a user's information, digital receipts could strengthen TrustBroker greatly. In the current system, a user never knows for sure what parts of his information are requested by a web server. Also, he never knows for sure what parts of his information were released by the repository. It may be that the web server requests and is granted access to much more of his information than is required to complete the transaction. Digital receipts could provide an audit mechanism to notify the user whenever his information was requested and/or transferred, in addition to which

server requested the information, and what credentials were used to satisfy the access control policies. This would allow the user to be aware of how his information is being distributed.

Second, the idea of context in trust negotiation is becoming a significant concern. The problems lies in a malicious remote entity's attempt to either phish for another's credentials or create a DoS attack. The first attack could be levied by submitting irrelevant policies in the attempt to collect information about which credentials the entity owns. The second attack could be effected by indefinitely continuing rounds of negotiation with irrelevant policies and credentials, thus consuming the entity's resources. Either attack could be avoided if the server understands the context of the request for information and matches it with the context of the policies and credentials. If a server receives policies or credentials that are out of context, they would simply be ignored. Although this might lead to a termination of the trust negotiation, it would help identify errors in policies and protect against malicious attempts to gather information or cause a DDoS situation.

Applying context to the trust negotiation that ensues between the web server and repository would strengthen TrustBroker. It would help ensure that the repository divulges only the information relevant to the current transaction.

References

[1] D. Eastlake and T. Goldstein. ECML v1.1: Field specifications for e-commerce. *IETF Request for Comments: 3106*, April 2001.

[2] Carl Ellison and Bruce Schneier. Ten risks of pki: What you're not being told about public key infrastructure. *Computer Security Journal*, XVI(1), 2000.

[3] Jennifer Golbeck and James Hendler. Inferring reputation on the semantic web. In *Proceedings of the 13th WWW Conference*, New York, NY USA, 2004. ACM Press.

[4] A. Josang. The right type of trust for distributed systems. In *Proceedings of the 1996 New Security Paradigms Workshop*, New York, NY, USA, 1996. ACM Press.

[5] Michael Koch. Global identity management to boost personalization. In *Proceedings of the Ninth Research Symposium on Emerging Electronic Markets*, Basel, Switzerland, 2002. Institute for Business Economics, University of Applied Sciences Basel.

[6] Karl Krukow and Mogens Nielsen. Towards a formal notion of trust. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Principles and Practices of Declarative Programming*, New York, NY, USA, 2003. ACM Press.

[7] Microsoft. Microsoft .net passport for businesses. http://www.microsoft.com/net/services/passport/business.asp, March 2003.

[8] Rob Murawski. Centralized directory services and accounts management project. In *Proceedings of the 28th SIG UCCS Conference on User Services*, New York, NY, USA, 2000. ACM Press.

[9] Birgit Pfitzmann and Michael Waidner. Federated identity-management protocols. In *Proceedings of the 11th Cambridge International Workshop on Security Protocols*, Berlin, Germany, 2004. Springer-Verlag.

[10] Liberty Alliance Project. Liberty alliance architecture overview version 1.1. http://www.projectliberty.org/specs/liberty-architecture-overview-v1.1.pdf, January 2003.

[11] R.Guha. Open rating systems. http://tap.stanford.edu/wot.pdf, 2003.

[12] Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. Trust management on the semantic web. In *Proceedings of the Second International Semantic Web Conference*, Berlin, Germany, 2003. Springer-Verlag.

[13] Slashdot. Slashdot meta-moderation. http://slashdot.org/faq/metamod.shtml, May 2003.

[14] Slashdot. Slashdot comments and moderation. http://slashdot.org/faq/commod.shtml, June 2004.

[15] Kerry Taylor and James Murty. Implementing role based access control for federated information systems on the web. In C. Johnson, P. Montague, and C. Steketee, editors, *Conferences in Research and Practice in Information Technology, Vol. 21*, Adelaide, Australia, 2003. Australasian Information Security Workshop 2003, Australian Computer Society, Inc.

[16] Marjorie Templeton, Herbert Henley, Edward Maros, and Darrel J. Van Buer. Interviso: Dealing with the comlexity of federated database access. In *The VLDB Journal - The International Journal on Very Large Data Bases, Volume 4, Issue 2*, pages 287–318, Secaucus, NJ, USA, April 1995. Springer-Verlag New York, Inc.

[17] W3C. The semantic web. http://www.w3.org/2001/sw/, July 2004.

[18] W. Winsborough, K. Seamons, and V. Jones. Negotiating disclosure of sensitive credentials. In *Proceedings of the Second Conference on Security in Communication Networks*, Amalfi, Italy, 1999.