Delegate: A Proxy Based Architecture for Secure Website Access from an Untrusted Machine*

Ravi Chandra Jammalamadaka[1], Timothy W. van der Horst[2]
Sharad Mehrotra[1], Kent E. Seamons[2], Nalini Venkasubramanian[1]
University of California, Irvine[1] Brigham Young University[2]
{rjammala,sharad,nalini}@ics.uci.edu {timv,seamons}@cs.byu.edu

Abstract

Performing sensitive online transactions using computers found in cybercafés and public libraries is risky. The untrusted nature of these machines creates a target rich environment. A simple keystroke logger, a common payload of many viruses, records and transmits the secret information (e.g., passwords, credit card numbers, PIN numbers) entered into these machines. In addition, sophisticated malware can hijack a user's authenticated session to perform unauthorized transactions masquerading as the user.

This paper presents Delegate, a proxy-based architecture that enables a user to access web sites without disclosing personal information to untrusted machines. Delegate enforces rules at the proxy to detect and prevent session hijacking. This architecture leverages users' trusted mobile devices, e.g., cell phones, and requires no modification to web servers or the untrusted machines. Delegate is designed to provide a balance between security and usability.

1 Introduction

The widespread acceptance of the Internet as a medium of doing business introduces new avenues for identity theft. Many people access the Internet using public computers that are not under their direct control, such as those available in cybercafés, public libraries, and universities. A compromised computer may be under complete control of an adversary who can then log keystrokes/click streams, snoop incoming and outgoing data, and take screen shots.

In New York, an adversary stole about 450 online banking passwords during a 2 year period by installing a keyboard sniffing program on public terminals at 13 different Kinkos locations in Manhattan [1]. An insider working in a computer lab at BYU installed spyware that collected private information from 600 students [6]. A similar instance occurred at Boston College [2].

While not widely observed, more sophisticated active attacks are possible. For example, an attacker could hijack or piggyback on user's session at an online store and make additional purchases. In the case of an online bank, the attacker could transfer money to a different account while the user checks his balance.

The attacks available on public computing terminals are not limited to the theft of information. Many times an attacker is not concerned with what can be *obtained* by an attack but what can be *destroyed*. For example, when the user is accessing his/her email, an attacker could delete email messages or send spam (or dirty messages) to everyone in the user's address book.

People continue to use untrusted public computers in spite of these risks. For a traveler, a computer at a cybercafé may be the only resource for his/her computing needs. A laptop owner may be forced to use a public computer due to a lack of network connectivity.

This paper presents the design of Delegate, an architecture to safeguard users against the threat of attack while using untrusted public computers. The goals of Delegate are to prevent an attacker from: 1) Stealing a user's secret information; 2) Destroying a user's personal information; and 3) Hijacking a user's session in order to perform unauthorized transactions at a web server. Delegate assumes the user has a cell-phone or other trusted personal communication device that can be used to communicate with the user without relying on the untrusted machine. The user can send and receive personal information on the cell phone as well as validate and approve sensitive transactions.

The remainder of the paper is organized as follows. Section 2 presents our threat model and architecture. Section 3 details its current implementation. Section 4 shows how Delegate addresses session hijacking. A security analysis of Delegate is in Section 5 and experimental results are re-

*This research was supported by funding from the National Science Foundation under grant no. CCR-0325951 and IIS-0220069, the prime cooperative agreement no. IIS-0331707, and The Regents of the University of California.

ported in Section 6. Section 7 discusses related work, and Section 8 provides conclusions and future work.

2 Architecture

Delegate is designed to address and mitigate the prevalent attacks that occur in the context untrusted machines, namely: keylogging, password sniffing, shoulder surfing, and session hijacking. The other goals of the Delegate architecture are:

- Authenticate users who access web service providers from untrusted machines without requiring the user to reveal any long-term secrets that an attacker can use to impersonate the user in the future.
- Detect and prevent session hijacking in order to stop malware from performing unauthorized transactions while the user is performing legitimate operations.
- Limit the scope of potential damage by reducing the attack surface while the user is accessing the web from an untrusted machine.
- Minimize the number of changes that are required by a web server or a user in order to deploy the system.
- Create a system that is easy to use. The design must strike an appropriate balance between security and usability. If user's find the system too burdensome, they will quickly grow tired of using it and turn it off.

The design of Delegate is illustrated in Figure 1. There are four components in the architecture: 1) Trusted proxy; 2) Web server; 3) Untrusted computer; and 4) Trusted, mobile device.

Untrusted computer For this paper, a machine is untrusted if it is not under the administrative control of the user. This includes any computer where software can be installed without the user's knowledge or permission. An untrusted machine under complete control of an adversary is unlimited in the kinds of attacks it can launch.

Trusted proxy The trusted proxy is under the complete administrative control of the user. Typically, it is a home or an office machine.[1] This proxy stores a user's secret information and acts as a middleman between an untrusted machine and a web server by filtering all incoming and outgoing traffic between them.

The proxy: 1) Authenticates the user via the mobile device when it receives a request to establish a secure session from an untrusted machine; 2) Intercepts the user's requests and inserts secrets when necessary; 3) Requests user validation of any potentially dangerous requests via a secure channel with the trusted modile device; and 4) Removes any sensitive information from a web server's response before forwarding it to the untrusted machine.

[1] In cases where organizations do not allow individual users to host servers, the proxy should be maintained by the organization itself.

Web server A web server communicates directly with the proxy and is unaware of the untrusted computer. Since each user operates their own trusted proxy, no modification to web servers is required in this system

Trusted mobile device The user must possess a trusted mobile device, e.g., a cell phone, in order to use Delegate. The trusted proxy contacts the user through this device to obtain authorization for requests from the untrusted computer. Delegate is also designed to minimize the resources required by trusted moble device. If the cell phone is lost or stolen, it must be easy for the user to quickly revoke the trust that the proxy has in the cell phone.

Although many mobile devices can directly access web sites and avoid the security risks associated with untrusted platforms, they lack many of the advantages of using a desktop machine, including: 1) A larger display; 2) Greater network bandwidth; 3) More memory and processing power; and 4) No dependancies on battery life.

Usage 1) The user's requests to the web server from an untrusted machine is routed through the trusted proxy. 2) The trusted proxy validates the request using the mobile phone. The proxy establishes a session, and rejects any requests that are outside the domain of the original request. 3) The proxy inserts secrets into subsequent requests that require them, seeking explicit permission from the user via the mobile device if necessary. 4) The proxy scrubs sensitive information from pages before they are returned. 5) When a request is determined to be dangerous, the proxy obtains explicit permission from the user to forward the request. 6) The user terminates the session with the trusted proxy to prevent an attacker from performing transactions after the user leaves the untrusted computer.

2.1 Design Alternatives

A principal motivation for the proxy design is to deploy Delegate without requiring any changes to web servers. This provides a convenient way to bootstrap the usage of this system. However, this may ultimately hinder broad adoption because not all users are in a position to run their own proxy. The proxy service could be offered by a trusted third party, however this third party would to be trusted to store and manage a user's sensitive information and would be an attractive target for attackers.

If web servers incorporated the functionality of the proxy, then many problems would go away. Sensitive information would be distributed among servers who are already trusted with this information. The benefits of Dele-

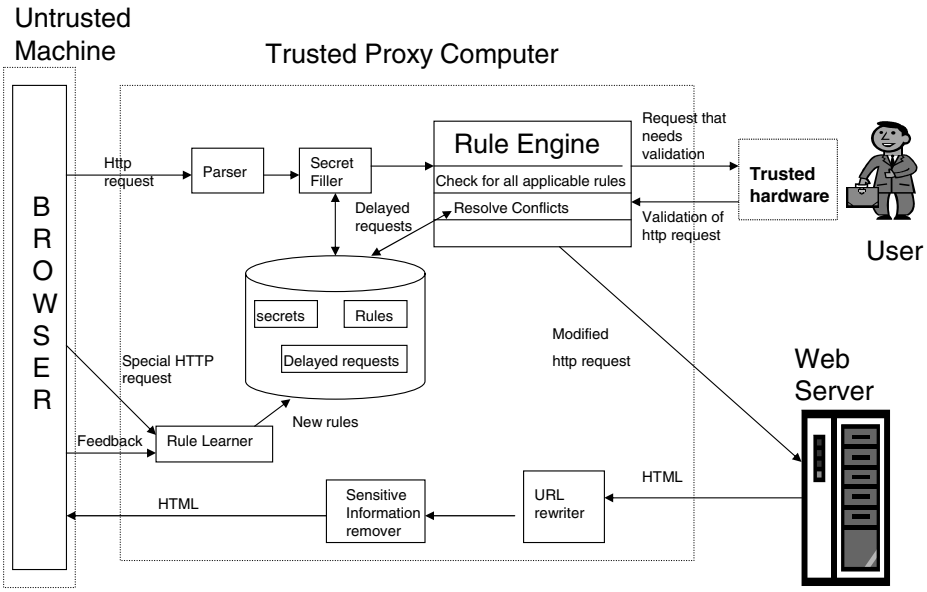

Figure 1. Delegate Proxy-Based Architecture

gate would be immediately available to a much larger audience than the proxy model supports, namely all users who have access to a cell phone with text messaging capabilities.

3 Implementation

We have implemented a preliminary Delegate prototype in Java. Messages are sent to a traditional cell phone via SMS text messaging or through a traditional socket to specialized software on the mobile device. This software is implemented in the Java micro edition environment (J2ME). Delegate enables access to websites running both HTTP and HTTPS, but for simplicity we refer to both as HTTP.

When the user requests to browse a website with SSL support, Delegate opens two SSL connections, one SSL connection between proxy and the untrusted public computer and another connection between proxy and the web server. After the SSL connections are made, the proxy can now act as a man-in-the-middle between the untrusted machine and the webserver.

Since an SSL connection is opened with the proxy, the certificate that the proxy produces during the SSL handshake is not trusted by the untrusted public computer. Even if the user is visiting a web site that he deems should contain a valid certificate, the user will be prompted with an invalid certificate alert message from the browser, since this certificate belongs to the proxy. To avoid such a situation, the user can purchase a valid certificate from a trusted certificate authority. If buying a certificate is not an option, then the user should carefully review and accept the certificate presented by the proxy.

Authentication To access a website through an untrusted computer, the user must first authenticate to the trusted proxy computer using a one-time password or PIN. Following an initial request, the proxy sends the domain name of the request along with this one-time PIN. The proxy also returns a login page to the untrusted machine where the user can enter the PIN. Alternatively, the user can send a confirmation back to the proxy via the cell phone. This requires an installation of our specialized software. Once the user is authenticated, the proxy is ready to accept requests from the remote untrusted computer to the website designated in the login process. The proxy opens a session with the web server and acts as the middleman, directing traffic between the untrusted public machine and the web server.

In order to reduce the attack surface, the authentication is for a specific domain. The proxy disregards any request that does not pertain to the website for which the proxy has opened a session. In some cases, a website can maintain images/multimedia objects in a different domain where the HTML files are hosted. Since the HTML files are funneled through the proxy before being sent to the untrusted machine, the proxy keeps track of all the URLs of the images/multimedia objects and allows the requests made to these objects. Any other request outside of this list and outside of the domain to which the user authenticated, is simply discarded. However, an adversary can send requests to the proxy pertaining to the website for which the proxy has opened a session. The proxy has to decide for every web

request forwarded, whether the request originated from the user or from the adversary. To do this, the proxy compares the requests against the set of rules provided by the user to determine when to contact the user for validation of a request.

Using our specialized software, the cell phone and the proxy cryptographically protect their communications. However, in many cases, the user may be willing to tolerate the risk that cell phone traffic is sent in the clear.

Secret Filling Essentially, there are two types of requests that are primarily used during web browsing, the HTTP GET and the HTTP POST. In general, GET requests are used to pull information from the web server and POST requests are used to send data to the web server. Secret filling by the proxy should therefore only required for POST messages. The following is an example of a POST message:

```
POST www.somehost.com/login.asp HTTP/1.0
User-Agent: HTTPTool/1.0
Content-Length: 32
Username=&password=
```

This POST message is generated when the user navigates to the login page of somehost.com and clicks the submit button without filling in the username and password information. When the proxy receives this request, the *parser* strips out the *data string* (the last line of this post message) and URL information, and passes it to the *secret filler* module. The data string contains a set of *key=value* pairs. A key is the metadata that is attached to every input element (textboxes, radio boxes, etc.) in an HTML form. In this example username and password are the two keys. The *secret filler* now fetches the required secrets from the local database using the URL information and the keys and fills in the secrets by modifying the data string. The secret filler module only fills in secrets that have been left blank by the user and leaves all other requests unmodified.

Validating HTTP requests The proxy must determine if a request it receives is *legitimate*, i.e., it is issued by the user and not malware on an untrusted machine. One way for the proxy to determine the legitimacy of a request is to obtain user validation via the cell phone. The method can be used to ask the user to validate *each and every* request the proxy receives. This solution does not scale and places an unbearable burden on the user.

Alternatively, the proxy can only require user validation for requests that are potentially *"damaging"*, i.e., the request involves a monetary transaction. Determining such kinds of requests is a non-trivial task and Delegate requires the help of the user in this regard. The user supplies the proxy with a set of rules. Using these rules, a *Rule engine* determines if contact with the user is necessary. These rules try to capture the *validation policy* of the user. In Section 4, we will show how these rules are captured and evaluated.

Removing sensitive information All responses from a web server are filtered by the proxy to remove any sensitive information before it they are forwarded to an untrusted computer. This is necessary because the response from a web server may contain sensitive information (e.g., credit card numbers, address, phone numbers) that would otherwise be exposed to the untrusted machine. Although many websites already obfuscate some of the sensitive information they display, e.g., only listing the last four digits of a credit card number, the proxy provides this functionality to all the web sites a user may visit.

The identification of what constitutes sensitive information in a certain context is a delicate task. We have identified several approaches that have thus far been effective. The simplest approach is to replace any secret that appears in the HTML response from the web server with "******". Although this method is useful, it does not deal with situations where the sensitive information cannot be predefined. For instance, when a user accesses an email which contains a new password (the attacker can force such an email) then the previous heuristic will not suffice. For these kind of situations, Delegate looks for sub-sentence patterns such as (Password: <any_word>) and hides them. Although both these approaches are useful, more work is needed to fully address this problem.

4 Session Hijacking

Delegate limits the potential malware has to hijack a user's session by fabricating or modifying dangerous requests to a web server. First, the user configures the browser on the untrusted machine to forward all web requests to the proxy. The user can rely on the proxy to authenticate to a web server and fill in all sensitive data. Delegate trusts the user to protect sensitive information and avoid entering it at the untrusted machine.

To prevent session hijacking, a simple solution is for the proxy to forward all requests to the user for validation. This solution has some severe drawbacks: 1) HTTP requests are not semantically rich enough for users to understand the purpose of the request; and 2) A typical web session can generate hundreds of web requests, placing a tremendous burden on the user. These factors make it probable that users will reject such a system as unusable or impatiently approve each request without evaluating it thoroughly. A successful solution must strike an effective balance between security and usability.

A significant number of web requests do not require user validation. For instance, requests to retrieve the images embedded in a web page that the user authorizes the proxy to

retrieve should not require further validation. The user is primarily concerned with dangerous requests that malware can exploit to cause harm to the user, e.g., an unauthorized request to transfer money from the user's account. Identifying dangerous requests is the key to addressing the session hijacking problem. Once a dangerous requests is identified, the user can be contacted for validation.

Determining the danger of a web request is non-trivial and subjective. Different users have different perceptions about what is considered dangerous. For instance, some users may feel that accessing their purchase history from Amazon.com should be classified as a dangerous operation, while others may not. Users have different *levels of paranoia*. Since users have their own view on what constitutes a dangerous request, a single strategy for determining dangerous requests for all users is undesirable. It would result in false positives and false negatives depending on the individual preferences of each user. For the system to be usable, it must be tailored to meet individual preferences for what constitutes a dangerous request.

In Delegate, users dictate *rules* to the proxy that identify dangerous requests. A rule specifies a set, or category, of requests and the appropriate action to take when it is encountered. Rules are an effective way to capture the user's *intent* in identifying dangerous requests. The set of rules constitutes the user's *policy*. Users specify a policy for every website they wish to access from an untrused machine. Once a dangerous request is identified, a text message that explains the semantics/purpose of the request needs to be sent to the user for validation. This task is difficult to handle automatically since HTTP request messages are not semantically rich. Delegate enlists the user's assistance to specify the contents of a validation message.

The remainder of this section discusses Delegate's model for classifying dangerous requests, capturing a user's policy, and rule generation.

4.1 Classification of Dangerous Requests

An analysis of popular websites in the areas of banking, travel, email, and retail revealed two kinds of dangerous requests are prevalent during the course of an online session.

Definition 1 - State Change Request (SCR): A request that alters the state of the web server permanently. Forwarding such requests to a web server without user validation has the potential for damage. For example, a request that transfers money from the user's account. A distinguishing characteristic of such requests is that they usually carry data supplied by the user in addition to the URL information.

Definition 2 - Private Information Request (PIR): A request that retrieves a user's private information from the web server. If the data is returned to the untrusted machine, it is potentially exposed to malware. In some cases, the information should be filtered out of the web page or returned to the user via the cell phone. In other cases, the user may allow it to be sent to the untrusted machine, but only when the user explicitly authorizes it. Unlike an SCR request, a PIR request does not have any noteworthy distinguishing characteristics to easily identify it.

Delegate assumes requests outside these two definitions are not dangerous.

4.2 Capturing a Policy

Policies in Delegate have a *request level* granularity. Although higher level policies may reduce the complexity of policy creation, they lose the flexibility to reduce the validation requirements. For example, assume a user specifies a policy that requests validation of PIR requests to a particular website. This may be overkill if the user is comfortable permitting some PIR requests without validation. Seeking validation from the user is a costly operation and hence should be avoided whenever possible if the security risk is acceptable. Hence, the policy language operates at the lowest granularity possible, at the request level.

Since validation is done at the request level, one might assume that it makes sense to group requests and validate them at the group level. However, this cannot be achieved since 1) A single request forwarded to the web server can do extensive damage; and 2) Most requests cannot be easily undone once they are completed.

The following is a formal definition for an HTTP request in Delegate policies:

Definition 3 - HTTP request object: An HTTP request R is modeled as a set of <attribute, value> pairs, denoted as R_{AV}. Each set contains at least two <attribute, value> pairs (i.e., $|R_{AV}| \geq 2$). The *URL pair* stores the URL information, e.g., <URL,"http://www.americanexpress.com">. The *Type pair* stores the request type, e.g., <Type,"GET">.

The other pairs in the set are dependent on the contents of the HTTP request and are application dependent. These pairs can be obtained from three different sources 1) the *query string* that is attached to the URL and follows the "?" sign, 2) the *data string* that is attached to every POST request, and 3) the *cookie string* that is attached to HTTP requests. These <attribute, value> pairs can be classified in two ways: 1) as *constant valued* pairs, or 2) as *random valued* pairs. Constant valued pairs are identical across all sessions, while the random values vary. The values of the constant valued attributes have some special meaning to the web server; they dictate the actions of the web server in response to the request. Random valued attributes provide session specific information such as session ids, session keys, etc.

The classification of <attribute,value> pairs as constant or random is based on a fundamental assumption regard-

HTTP_REQUEST:
GET /myca/onlinepayment/us/action?request_type=authreg_CardPayments
Accept: image/gif, image/x-xbitmap,
Referer: https://www99.americanexpress.com/myca/
Connection: Keep-Alive
User-Agent: Mozilla/4.0
Host: www99.americanexpress.com
Cookie: Manage=cards; s_session_id=1231122950204954-05-05

HTTP_REQUEST object:
<URL,/myca/onlinepayment/us/action> , Constant
<Type, GET>, Constant
<request_type, authreg_CardPayments >, Constant
<Mange, cards>, Constant
<s_session_id,1231122950204954-05-05>, Random

Figure 2. An example HTTP request and its corresponding HTTP request object

HTTP_REQUEST:
GET /myca/onlinepayment/us/action?
request_type=authreg_acctAccountSummary
Accept: image/gif, image/x-xbitmap,
Referer: https://www99.americanexpress.com/myca/
Connection: Keep-Alive
User-Agent: Mozilla/4.0
Host: www99.americanexpress.com
Cookie: Manage=cards; s_session_id=1231122950204954-05-05

Figure 3. HTTP GET request generated when accessing account summary from americanexpress.com

ing the design methodology of websites. It assumes that the actions that are triggered at a web server in response to a request only depend on the values of the constant valued attributes. Therefore, two requests with the same set of constant valued attributes names but different values can have different semantics at the server side. This assumes that the random valued attributes do not dictate semantics at the web server side. This is a reasonable assumption, since it is unclear why websites would be designed in such fashion. We have manually validated this assumption by examining the request messages for many popular websites and we have not encountered any website that contradicts this assumption.

Also, an HTTP request contains some header information such as accept, referer, etc. Delegate ignores these attributes because any alteration of these values by malware can only cause denial of service and are not significant for protecting against dangerous requests.

The formal model of an HTTP request message as an HTTP request object prunes away some unnecessary information in the request and classifies the rest of the information as constant or random. Fig 2 shows an example HTTP request and its corresponding HTTP request object. This example also includes the classification of the <attribute, value> pairs. This request is generated when a user is trying to access his/her credit card payment history from americanexpress.com. Only a portion of the HTTP request is shown for simplicity.

HTTP_Template:
<URL,/myca/onlinepayment/us/action>, constant
<Type, GET> , constant
<request_type, authreg_CardPayments >, constant
<Manage, cards>, constant
<s_session_id, *>, Random
Proxy Action:
Validate
Message:
Trying to access the American express account summary.

Figure 4. An example Rule

A *rule* is the basic building block of the policy language. Each rule instructs the proxy which action to take for certain kinds of requests. Each rule has the following structure:

RULE:
<HTTP_TEMPLATE> <PROXY_ACTION> <MESSAGE>

HTTP_TEMPLATE describes the class of requests pertaining to the rule. Informally, an HTML_TEMPLATE has to encompass a set of HTTP requests. PROXY_ACTION describes the action that the proxy takes for the class of requests, and MESSAGE contains the message the proxy sends to the user when validation is required.

Definition 4 - HTTP TEMPLATE: An HTTP Template contains a set of <attribute,value> pairs, denoted as T_{AV}. The cardinality of the set T_{AV} is at least two. The HTTP template also contains the URL and the type pairs. The rest of the <attribute,value> pairs are classified as either constant or random. For constant valued attributes, the value is stored in the rule. For random valued attributes, no value is stored.

The following definition describes when a request R conforms to a HTTP Template T.

Definition 5 - Template Conformance: A request R conforms to an HTTP Template T, when the following holds:

- R.urlpair = T.urlpair
- R.typepair = T.typepair
- Let all constant valued <attributes,value> pairs in R_{AV} be represented by the set R^C_{AV}. Let all constant valued <attributes,value> pairs in T_{AV} be represented by the set T^C_{AV}. Then, $R^C_{AV} = T^C_{AV}$.
- Let all random valued attribute names in R_{AV} be represented by the set R^R_A. Let all random valued attribute names in T_{AV} be represented by the set T^R_A. Then, $R^R_A = T^R_A$

Fig 4 shows an HTTP_TEMPLATE example. A request R conforms to a Template T, if all constant valued pairs from R match their corresponding pairs in T. The random variable values are ignored since they are session specific.

Consider the two HTTP GET requests in Fig 2 and Fig 3. The GET request in fig 2 is generated when the user is trying to access his/her payment history from amer-

HTTP POST MESSAGE:

Accept: image/gif,
Referer: http://home.americanexpress.com/home/mt_personal_cm.shtml
Connection: Keep-Alive
User-Agent: Mozilla/4.0

UserID=bobwiley009&Password=notherealpassword&manage=cards

Figure 5. An example POST message

Validation Message:

Are you trying to log in to americanexpress.com??
Form reconstruction:
UserID = NOT_SPECIFIED_TO_BE_FILLED_IN_BY_THE_PROXY
Password = NOT_SPECIFIED_TO_BE_FILLED_IN_BY_THE_PROXY
NO_DESCRIPTION=cards

Figure 6. An example Validation Message

icanexpress.com, while the other GET request is generated when the user is trying to access his account summary from the same website. Both these requests have the same set of constant valued attributes, but the value of the action_request_type dictates the information that the web server should return.

PROXY ACTION: PROXY_ACTION specifies the action the proxy takes when an HTTP request conforms to the HTTP_TEMPLATE. There are four possible proxy actions:

1. **Validate:** The proxy contacts the user via the cell phone to validate the HTTP request.
2. **Accept:** The proxy forwards the HTTP request to the web server without any user validation.
3. **Defer:** The proxy stores the HTTP request locally and not forward the request to the web server. The user can validate the request once he/she is back working on the trusted proxy. *Defer* action is very helpful, since it saves the bother of contacting the user via the cell phone. Such an action is applied to requests which are not urgent. Since HTTP is a stateless protocol, requests can be stored locally and sent at a later time during a new session to have the same effect.
4. **Drop:** The proxy drops the request and does not forward it to the web server. This applies to requests that the user never intends to issue from an untrusted machine.

MESSAGE: MESSAGE refers to a text message that is sent to the user when a request sent by the user satisfies the HTTP_TEMPLATE and PROXY_ACTION is set to *Validate*. Otherwise, the *MESSAGE* is left blank. The message states the semantic meaning or the purpose of the request. Since this message is given by the user, the user can easily identify the essence of the request when the message is received on the cell phone.

Validation messages need to be sent to the user only for HTTP GET and HTTP POST messages. Even though there are 6 other types of HTTP messages, these are the only two relevant to the Delegate architecture. An HTTP GET message retrieves a resource from the web server, while an HTTP POST messages sends data to a web page for further processing. Fig 2 shows an example of the GET request and figure 5 shows an example of the HTTP POST request. The end of a POST request contains a *data string*, which contains the data that is being submitted to the web server.

To validate an HTTP GET message, the entire message is sent to the user. Validation of HTTP POST messages is not that simple. Typically, POST messages are generated when a user submits a form. When a post message is received, the proxy constructs a text representation of the form and sends it to the user for validation. A text representation of a form is constructed in the following manner: a) The descriptive text preceding each input element is located. b) The set of <descriptive_text, input_value> pairs for all the input elements constitutes the text representation of the form. Fig 6 shows the validation message sent to the user when the proxy receives the POST message in fig 5. Note, the value *cards* does not have any descriptive text in the form.

Improving the expressiveness of the HTTP templates Previously, we have defined HTTP templates that are applicable only for a particular kind of requests. To provide set semantics to HTTP Templates, we allow regular expressions to be placed in the value part of the <attribute, value> pairs. We denote such templates as *coarse templates*. For such templates, we modify Definition 5 as follows:

Definition 5b: A request R conforms to a coarse template T_C, when the following holds:

- R.urlpair matches with the regular expression specified in $T_C.urlpair$.
- R.typepair matches with the regular expression specified in $T_C.typepair$.

4.3 Conflicting Rules

The rule-based approach is reminiscent of firewalls. The proxy acts as a remote, personal firewall to protect the user's privacy and security. As in traditional firewalls, a single request can conform to several templates with conflicting actions. For instance, consider a user providing his validation policy to the proxy for the amazon.com website. The user does not want to validate any GET requests unless the GET request fetches his/her past purchases. The easiest way to create this policy is to create two rules, one rule accepts all GET requests and another rule validates any GET request for the history page. These two rules clearly conflict. If conflicts were not allowed, the user is forced to specify a rule for every web page that she expects to access.

The actions in Delegate have a clear precedence: Validate, Defer, Drop, and Accept. When two different rules conflict, the conflict is resolved according to the precedence order. The *conflict resolution operation* is associa-

tive, therefore, when more than two rules conflict, the order in which the conflicts are resolved does not matter.

4.4 Rule Generation

Delegate relies on a policy database of rules for each website the user accesses from an untrusted machine. Policies can originate from someone other than the user. For instance, security experts can develop policy files for certain websites that a user can easily install and use. A user could customize the policy to meet her requirements. Another likely scenario is for a webmaster to provide policies that identify dangerous actions and provide meaningful validation messages to the user. The webmaster is well-suited to understand those requests that pose the greatest threat to users if exploited by malware. A webmasters could specify a range of policy options geared for users with different levels of privacy preferences. As the Delegate architecture is deployed, websites would have a competitive advantage by providing their users with protective policies when using untrusted machines. The system is more suitable for non-technical users if experts supply the policies.

We are exploring an approach to policy generation that can be used by experts to easily create policies. This approach has the potential to make it easier for non-experts to generate their own rules. The approach is to access the web from the trusted machine that runs the proxy, and have the proxy assist in automatically generating rules based on user feedback. To achieve this, the user first switches Delegate into *learning mode* and then starts to access websites as he/she normally would through a proxy.

Whenever the proxy encounters an SCR or a PIR request, Delegate asks the user whether a rule should be generated. Since if is difficult to determine a PIR request compared to an SCR request, Delegate assumes every request to an image or multimedia object is not a PIR request. Thus, requests that access web pages (i.e. requests that access html files, asp pages, etc) are assumed to be PIR requests. For instance, assume that the user is accessing his email account and has just created an email and pressed the send button. This generates a POST message that is forwarded to the proxy. The proxy now asks the user if he wants to create a rule for this action. If the user responds affirmatively, the proxy creates an HTTP_TEMPLATE for the request as described in the previous sections. Now the user can modify the default template, by placing the "*" operator and making the HTTP TEMPLATE more generic. Since the user is fully aware of this last action (in this case, the user sent an email), the user chooses an appropriate proxy action and creates a meaningful message to complete the creation of the rule if validation is required. Initially, the rule is stored in the policy database and is not immediately applicable for future requests. Since the proxy needs to classify the <attribute, value> pairs of the HTTP Template as either constant or random, the proxy waits until the next time it encounters the same request in a different session so that it can check for constants. After receiving the rule twice, the proxy then can apply the rule with immediate effect.

Training the proxy could require a significant effort on the part of the user, since the policies are website specific and users wish to access them from the untrusted machine. More time spent training the proxy may significantly reduce the number of attempts to contact the user via the cell phone for request validation. Thus, the better the proxy is trained the easier it should be to use Delegate because false positives and false negatives are reduced.

5 Security Analysis

The use of a completely untrusted machine provides limitless possibilities for attack. As such, browsing and accessing web services from an untrusted device will never be as secure as using a trusted device. Delegate significantly reduces the attack surface in order to achieve an acceptable level of risk.

Using a keystroke logger or by shoulder surfing, an attacker cannot acquire secrets belonging to the user because they are never entered into the untrusted machine. Also, an attacker cannot sniff for passwords or other secrets on the local network since the secrets never traverse the network used by the untrusted machines. An attacker will, however, be successful in procuring any information entered at the untrusted machine.

The cell phone is an attractive target for attack because it has the ability to receive the one-time PINs from the proxy. If the cell phone is lost or stolen, any previously received PINs that remain on the phone are worthless. However an attacker could successfully receive one-time PINs until the proxy is informed that the phone has been compromised.

The proxy contains user secrets. To mitigate the effects of a compromise, the proxy should: 1) Run in its own user space; 2) Keep an audit log; 3) Refuse connections/requests from any machine other than the one that has supplied the one-time PIN; and 4) Only allow one active connection.

Web servers also store user secrets which may be accessible to the untrusted machine through two methods. First, the web server may disclose this information on a web page sent to the user, either visibly (e.g., the page the user sees has the info on it) or hidden (e.g., a hidden form value). Second, if an attacker asks a web site to send an email-based password reminder the password, or a link to change it, will be exposed to the untrusted machine when the user access her email account. Filtering at the proxy should be able to counter both these attacks, provided the attack relies completely on the proxy for information on the user.

Without a successful active attack on the cell phone,

proxy, or web server, the untrusted machine only has access to the information that the proxy releases and what it can glean from publicly available information on the Internet.

Communication Links No security is guaranteed for communication between the untrusted machine and the proxy because it is in complete control of the untrusted device. No assumptions as to the integrity or confidentiality of the messages passed in either direction of this link should be made. Even though it provides no guarantees, a TLS connection should be used on this link to avoid passive eavesdropping by others on the network. The presentation of a valid one-time PIN means that the user has given authorization to the possessor to forward HTTP requests to the proxy and that the cell phone is ready to participate in the authorization of those requests. All that attackers should be able to gain by acquiring a valid PIN is that they can make HTTP requests to a specific website via the proxy.

The unidirectional link between the proxy and the cell phone should provide integrity, confidentiality, and authentication. However this is not a feasible requirement to enforce for all mobile devices, especially low-end cell phones. The short-lived nature of the one-time PINS removes some of the risks of using an insecure connection in this link.

The link between the web server and proxy is controlled by the proxy. It has the same assurances as any link can between a trusted machine and a known web server.

The untrusted machine can communicating directly with the web server, but by withholding the necessary authentication credentials from the untrusted machine prevents it from impersonating the user.

The untrusted device could attempt to make a communicate directly with the cell phone. These attempts cannot be prevented, but the user should be alerted if they occur.

Although accessing the website directly through the cell phone offers great protection to the user's personal data, the user loses out on the large screen, full-sized keyboard, bandwidth, and processing offered by the untrusted device.

Other Considerations This system is designed so that all messages must funnel through the proxy, providing a central location to audit all activity. The only way to bypass this auditing system is for malware on the untrusted computer to establish a direct link with a web service. This should only be feasible if an attacker has obtained the required authentication credentials. Auditing information recorded after a compromise of the proxy cannot be trusted.

The security of this system relies on the ability to a user to recognize invalid requests and not authorize those requests. Messages sent to the cell phone should be clear and understandable because the user must be able to make an informed decision based on their contents.

Blocking all outgoing traffic on the untrusted machine causes a denial of service. A user may interpret this as a normal technology failure and not an attack, and could simply access a website directly and disclose sensitive information simply to complete a task. Preventing this type of denial of service is outside of scope of this paper, however users must be educated in order to avoid this scenario.

By making superfluous requests, an attacker could force the proxy to send a large amount of validation messages to the user's cell phone. In Delegate, the user may disable the current proxy session via the cell phone to end this frustrating attack. However, the threat of an attacker bombarding a cell phone with messages remains as there are web and email interfaces for sending text messages to a cell phone at no cost to the attacker.

6 Experimental Results

This section describes several experiments we conducted with the Delegate prototype to measure the overhead introduced by the Delegate system and the benefit of using a policy based approach to control session hijacking. A full version of the paper provides more details[11]. The Java-based proxy was running on a Pentium 4 machine with 512 MB ram and Windows XP. The cell phone used is a Sony Ericsson w800i capable of running J2ME applications.

In the first experiment we measured the delay in mean response time for an HTTP request made from the untrusted public machine that uses Delegate. Delays are due to a) The proxy intercepting the HTTP requests and filling in the secrets; b) Message transfer between the proxy and the cell phone; and c) The validation process by the user. It took less than 1 msec for the proxy to intercept HTTP requests and fill in the relevant secrets. This time includes the parsing of the HTTP request, secret retrieving and reconstruction of the HTTP request. Since the proxy runs on a fairly powerful workstation, the secret filling time is negligible. Transferring of a message of 200 bytes using the specialized software installed in the cell phone took on an average 420 msecs. Since most messages are less than 200 bytes, 420 msecs could be regarded as the average transfer time of messages from the proxy to the cell phone. When the same message was transferred via SMS text messaging, the average delay was 4 secs. The biggest delay is caused when the user validates a messages sent from the proxy. The user has to carefully read the messages and understand them, before validating them. Since the user's mind is occupied in this process, he/she is unlikely to notice this delay.

In the second experiment we measured the benefit of a policy based approach to control session hijacking. In this method, requests known to be dangerous are identified in a policy. Since PIR requests do not have any distinguishing characteristics, we assume requests accessing images, mul-

timedia objects, etc., are not PIR requests. Without a policy, the proxy considers all SCR and PIR requests as dangerous.

The two important lessons learned from this experiment are a) a policy based approach significantly reduces the validation requirements when compared with automatic approaches, and b) the average number of validation messages per session for a typical user is 3.7. In contrast, when no policy file is present the average number of validation messages per session is 23.1. These preliminary results illustrate that a trained proxy with site-specific policies can significantly reduce the validation burden on the user. This is important because it is unlikely that user's will tolerate excessive message validation via the cell phone. An open problem is to identify characteristics that might allow an untrained proxy to better automatically identify requests that users consider dangerous, which would reduce the need for users or experts to generate policies in advance.

7 Related Work

Ross et al. [8] created a composable framework for multi-modal access to content using both trusted and untrusted devices. Their system specifies a proxy-based approach for inserting secrets into requests, scrubbing sensitive data from responses, and validating dangerous requests through a trusted device. The framework is broad, general-purpose, and extensible. No low-level details were given for the rule language used to identify dangerous requests and not all of the proposed features were implemented.

Wu et al. [9, 10] present a high-level overview of the proxy-based architecture similar to the one used in this paper. They also provide valuable insights into how actual users interact with such a system. They do not provide a solution for session hijacking.

Opera et al. [5] created a system that uses VNC [7] to connect an untrusted machine to a trusted machine. The untrusted machine receives short-lived, read-only access to the trusted machine. This system is not vulnerable to session hijacking because all input events must originate from a trusted mobile device and must arrive via a separate, secure communications channel. Due to the constraints of the mobile device it may be difficult and tedious to use it as the sole means of input.

SpyBlock [3] and Vault [4] use virtual machines to provide a sandbox for potential untrustworthy software, e.g., a web browser. Although the trusted and untrusted components operate on the same machine, these projects cover many of the same topics addressed by this paper. SpyBlock can prompt the user to validate potentially harmful requests and provides an API for servers to identify potentially dangerous actions.

8 Conclusions

This paper presented Delegate, a proxy-based architecture that allows users to access web services from an untrusted machine without revealing their sensitive information to that untrusted platform. The proxy acts as a middleman and filters all the traffic between the untrusted machine and the web service provider to insert secrets on the user's behalf. Delegate does not require any changes to the web server's authentication protocol. Delegate also permits the user to define a rule base that specifies whether requests are dangerous or not, and the proxy can contact the user via a cell phone to gain approval to submit requests that pose risks if exploited by malware.

The ideas in the Delegate design can be incorporated into Web sites directly, which would make them suitable for users on trusted and untrusted machines. Our design is targeted at cell phone users with text messaging capabilities, so the pool of potential users is very high.

The primary contribution of the paper is the specification of a policy language for establishing rules to classify dangerous Web requests and the appropriate action. This methodology has similarities to firewall rules that dictate Internet traffic that can pass through a firewall and traffic that should be blocked. The Delegate proxy is a type of remote personal firewall that inspects traffic and determines whether it is permitted to flow to or from a Web site and provides a layer of defense between the untrusted machine and a website.

References

[1] http://www.securityfocus.com/news/6447

[2] http://news.com.com/2100-1023-983717.html

[3] C. Jackson, D. Boneh, and J. Mitchell. Spyware Resistant Web Authentication using Virtual Machines. http://crypto.stanford.edu/spyblock/spyblock.pdf, 2006.

[4] P. Kwan and G. Durfee. Vault: Practical Uses of Virtual Machines for Protection of Sensitive User Data. PARC Technical Report.

[5] Oprea, D. Balfanz, G. Durfee, and D. K. Smetters. Securing a Remote Terminal Application with a Mobile Trusted Device. In proceedings Annual Computer Security Applications Conference (ACSAC 2004), Tucson, AZ, December 2004.

[6] http://deseretnews.com/dn/view/0,1249,600154978,00.html

[7] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. IEEE Internet Computing, 1998.

[8] S. Ross, J. Hill, M. Chen, A. Joseph, D. Culler, E. Brewer. A Composable Framework for Secure Multi-Modal Access to Internet Services from Post-PC Devices, IEEE Workshop on Mobile Computing Systems and Applications, December 2000.

[9] M.Wu, S.L.Garfinkel, R.Miller (2003) Secure Web Authentication with Mobile Phones MIT Student Oxygen Workshop.

[10] M.Wu, S.L. Garfinkel, R.Miller. Short talk: Secure Web Authentication with Mobile Phones DIMACS Workshop on Usable Privacy and Security Software, 2004.

[11] R.Jammalamadaka, T. van der Horst, S.Mehrotra, K.Seamons, N.Venkatasubramanian. Delegate: A Proxy Based Architecture for Secure Website Access from an Untrusted Machine. Technical Report No: TR-Rescue-06-13.