Implementing Security Policies Using the Safe Areas of Computation Approach

André L. M. dos Santos and Richard A. Kemmerer
Reliable Software Group
Computer Science Department
University of California
Santa Barbara, CA 93106 USA
{andre, kemm}@cs.ucsb.edu

Abstract

The World Wide Web is playing a major role in reducing business costs and in providing convenience to users. Digital Libraries capitalize on this technology to distribute documents that are stored in their servers. Online banks capitalize on this technology to reduce their operating costs and to offer 24 hours services to their clients. These two services are examples of services that require a high degree of security. Therefore, they require a higher level of protection than the existing technologies commonly used in the World Wide Web.

An approach that can be used to protect Internet transactions, called Safe Areas of Computation, was described in [DK99]. This paper describes the access control lists used by the Safe Areas of Computation approach, the operations on these access control lists supported by the approach, and how the access control lists can be customized for implementing many different security policies. This paper also describes example policies that can be used to protect Digital Libraries and Online Bank services. The paper uses these two services as an example of how the generic security policies supported by the SAC approach can be composed.

1 Introduction

Currently the computer systems and software used by the average user offer virtually no security. In addition, the Internet and the World Wide Web (WWW) have had a phenomenal growth during the past few years, interconnecting average users and connecting the average user to expert users that are not always good neighbors. Because of this many attacks, both simulated and real, have been described by the security community and have appeared in the popular press [HOP96, DFW96, DDK97]. Some World Wide Web services, like virtual banking, require a stronger degree of security and can be negatively impacted if their services are tampered with or used in a malicious way. An approach called Safe Areas of Computation (SAC), which was described in [DK99], can be used to provide the additional security required by these services. It uses trusted devices, such as smart cards, to provide an area for secure processing and storage. The Safe Areas of Computation approach is used in a client server configuration, where client Safe Areas of Computation communicate with a server Safe Area of Computation to perform authentication and to established a secure channel, which is a logical encrypted channel that uses the triple DES algorithm [NIS99] with a 168 bit session key. A simplified representation of an implementation of the SAC approach used for protecting Internet transactions is shown in Figure 1.

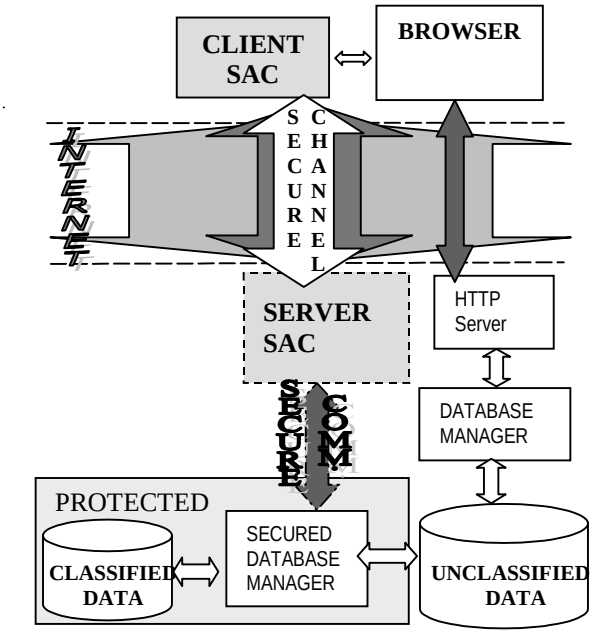

Figure 1: Safe Internet transactions using the SAC approach

When the SAC approach is used to protect Internet transactions, each client has a client SAC, which holds its access control list and other security relevant information, and each server has a server SAC. There are two different types of server sites: central authority sites and service provider sites. Initially there is a single central authority site with which the client SAC can communicate. A central authority site is the only site that can give a client SAC additional access to other sites. However, once a

service provider site has been added to a client site, Internet transactions are performed between the service provider site and the user without intervention from a central authority. A service provider can also modify the portion of a client SAC access control list that refers to that particular service provider, and it can remove all of the user's access to the site. It cannot, however, modify the user's access lists for any other site.

The Safe Areas of Computation approach uses the access control lists stored on the client SAC to enforce security policies. In addition, the approach uses operations on these access control lists to perform the necessary update actions.

In the SAC approach data is organized in multi-level security containers. The container model used is a variation of the container model described in [LAND84], which was proposed for military multi-level security documents, where each container is an abstraction for a set of data that has some attribute in common. The SAC approach uses pointers to data and headers to implement its container model. A header is a set of zero or more metadata entries that are used in the process of searching for specific data.

This paper has four additional sections. The next section describes the access control lists used for the approach and the operations that are allowed on these access control lists. The following section describes how the access control lists can be used to enforce different security policies. The paper then presents two test beds that use the SAC approach to enforce specific security policies and ends with conclusions and future work.

2 SAC Access Control Lists

The SAC approach uses three types of access control lists (ACLs) to implement various security policies based on the security required to access the data and on the clearance of the user. These ACLs are stored in the client SAC, and this is where the checks are made. The three access control list types are simple, hierarchical and complex.

The simple ACL contains records with one field. When an access to a data item requires a simple clearance, the client system presents the required clearance to the client SAC with the request for data. The client SAC checks its simple ACL for the necessary clearance (a one-byte number), and the data will be transmitted or decrypted only if the user has the required clearance.

The simple ACL has the ability to easily give a yes or no answer. However, in many cases, it is desirable to use hierarchical rather than absolute security clearances. This is the case, for example, when hierarchical clearances, such as classified, confidential, secret and top-secret, are used. That is, a top-secret clearance could access any document. A secret clearance could access secret, confidential and classified documents, and so on. The hierarchical ACL is used to implement this type of policy. Each hierarchical ACL record has two fields, representing a low and a high boundary. The security attribute of a data item that requires a hierarchical clearance contains the required clearance (a number), and the client SAC checks each record of its hierarchical ACL against the clearance. Access to the data is granted if the necessary clearance lies between the low and high boundary of one of the associated ACL records. For example, a hierarchical ACL could have the entries shown in Table 1, where each entry has the value that would actually be in the ACL, and the meaning of that value is given in parenthesis. In this example, data with a security attribute of 20 (<navy, secret>) can be accessed, while data with a security attribute of 87 (<nuclear, top-secret>) can not be accessed by this particular user.

Low	High
2 (<army, classified>)	4 (<army, secret>)
18 (<navy, classified>)	21 (<navy, top-secret>)
84 (<nuclear, classified>)	86 (<nuclear, secret>)

Table 1: Example hierarchical ACL

The two ACLs described above are used to represent static permissions. It is also possible for the server to request changes to the ACL, but this is assumed to occur infrequently. For permissions that change frequently and for more complex security requirements complex ACLs are used. An example of this type of permission is allowing a user to access documents only a certain number of times per day. Each record of a complex ACL has four fields that represent:

- Security Label. This is a yes or no field that represents the clearance of the user. The first check the client SAC makes on the complex ACL is to search for a record that has the required security label. The access is denied if there is no such record.
- Reference. This is a field that is used as reference for changes. It is this field that indicates what hour, day, year, concert, etc… is being considered to make a decision.
- Tokens Remaining. This is the field that is used to store the data that is going to vary frequently (often a counter). The basis for a decision of granting or denying access to data will be in this field in most cases.
- Reset Value. This field is a value that is transferred to the "tokens remaining" field when some particular actions occur, for example the reference value is updated.

An example of a complex ACL is shown in Table 2. The reference field in Table 2 represents the day of the year of the last operation on this field. Thus, the first entry represents a complex ACL that enforces the policy that a

user can transfer up to $300.00 per day. The first entry shows that the user can perform transfers, that he/she last performed a transfer on September 9th, that he/she still has 10 tokens ($100.00) that can be transferred on September 9th, and that if the current day is later than September 9th, then the tokens remaining field should be reset to 30 ($300.00).

Security Label	Reference	Tokens Remaining	Reset Value
2 (transfer)	252	10	30
8 (buy stock)	12	20	30

Table 2: Example complex ACL

In addition to accessing the ACLs when validating a request, the Safe Areas of Computation approach accesses the ACLs when the server requests updates. These update operations are performed at the beginning of a session, just after the establishment of the session key. A session is only considered valid after the client SAC acknowledges the requested updates. The next subsections describe both the operations that can be performed when requesting a data item and the operations for updating the access control lists.

2.1 Validation of Data Item Request Operations

The security attribute of a header or data item includes information that describes which ACL the client SAC has to check to make a decision. In addition, for complex ACLs this value describes an operation to be performed. The possible operations are shown in Table 3. In some cases when a complex ACL is required, another value called the comparison value is also sent to the client SAC. The comparison value is used for making a decision, such as what the current date or time is.

Operation
Add a value to the "tokens remaining" field
Subtract a value from the "tokens remaining" field
Check if comparison value is the same as "reference" field
Check if the "tokens remaining" field is positive.
Check if comparison value is the same as "reference" field. Copy the "reset value" field into the "tokens remaining" field if they are not the same.
Check if comparison value is lower than "tokens remaining" field.
Check if comparison value is greater than "tokens remaining" field.
Check if comparison value is lower or equal to "tokens remaining" field. If it is, subtract it from "tokens remaining" field and update the field for this value

Table 3: Complex ACL access control list operations

Some of the operations performed on complex ACLs that are shown in Table 3 will change the "reference" field and/or the "tokens remaining" fields. It would not be safe to trust requests from the client platform to decide whether to commit these changes, since a malicious client could exploit the possibility to change the value of one or both of these fields. On the other hand, the SAC approach may need to use the updated value to decide whether or not to contact the server, and whether or not to deny the operation on the client side. To address this problem, the client SAC uses a temporary variable to hold the updated value, and it bases its decision to contact the server on the value of this temporary variable. If the server SAC approves the operation the temporary variable is saved as a field of the complex ACL record.

An additional threat when using a complex ACL is present when an addition operation is requested. As discussed above, the addition is committed only as a result of a server confirmation. This could be misused for replay attacks during a session, since the session key would be fixed. In order to prevent this, the server sends an additional 24 random bytes when confirming an addition operation. The "xor" operation is applied to these 24 bytes and the session key to produce a new session key, which is used from this point on. The same scheme could be used to prevent replay attacks against other operations that change fields in a complex ACL entry.

2.2 Access Control List Update Operations

The Server SAC may request an update of the client SAC immediately after mutual authentication. The request is encrypted using the session key and a session is not considered valid until the server has received a message from the Client SAC confirming the update. This update may result in the addition of service providers or a modification of the ACLs corresponding to the service provider involved in the authentication. In addition to these updates, special service providers can also invalidate a client SAC.

The server SAC checks a database for the necessary updates of a user's ACLs before sending an authentication answer message to the user. This database is a set of files accessible to the server SAC, with each file being named as the user's ID and an extension "upd". Each update is encoded in a line of the update file as an operation code and a variable number of parameters. The server SAC encodes the number of updates that are going to be requested inside the authentication answer message. Following the authentication answer message the server SAC sends all the necessary updates as a stream of update codes and parameters, encrypted with the session key.

The client SAC also implements a number of updating operations that make it possible to implement many different policies. These operations are:

1. General operations: delete ACLs and add service provider.
2. Operations on simple ACL: add an entry, delete an entry, add a range of entries, and delete a range of entries.
3. Operations on hierarchical ACL: add a range and delete a range.
4. Operations on complex ACL: add a label, delete a label, delete a range of labels, entry expiration by reference, token expiration by reference, and token reset by reference.

3 Security Policies

The SAC approach is very flexible and can enforce many different policies. These policies may be used to restrict access to data based on an absolute or relative security label, or on parameters, like number of times that data from a certain group can be accessed, dates and/or time of access. All of the different possibilities are too numerous to be discussed in this paper, but in this section some basic policies that may be combined to express other, possibly more complex, policies are presented.

Some of the policy descriptions refer to changing or comparing values from outside the client SAC to values in the client SAC access control list. This does not represent a security threat because the value outside the client SAC that is compared or used to change a value either comes from the server SAC through a secure channel or it will be validated by the server SAC before being sent.

3.1 Simple Access Policy

The simple access policy is responsible for enforcing that a user can only access data that he/she is cleared for. The access clearance may be based on an absolute clearance, where the user either has the clearance or not, or on a relative clearance, where clearances are hierarchical.

The absolute clearance is easily expressed using the simple access control list. The server site assigns a value for each possible clearance and a client SAC either has the clearance or not. The client SAC checks the simple ACL for the required clearance when requesting data or containers.

The relative clearance is easily expressed using the hierarchical access control list. The server site divides the possible clearances in groups where an ordering has the obvious meaning. In addition, the server assigns a hierarchical value for each possible clearance. A user has entries in the hierarchical ACL that express his/her clearance for each group, and the client SAC checks the hierarchical ACL for the required clearance when requesting data or a container.

3.2 Simple Counter Policy

The simple counter policy is responsible for enforcing the restrictions that a user may access data that he/she is cleared for only a certain number of times. This policy may also require the server to be able to add more usage privileges to a user.

This policy can be implemented using the complex access control list. The security label field of the complex ACL represents whether the user is cleared to access the data. The tokens remaining field represents the number of times the user is still allowed to access the data. One token is subtracted from the tokens remaining field every time data is accessed. This is accomplished through the "subtract from tokens remaining" operation defined in Table 3. Thus, in order to check if the tokens remaining field is greater than zero and if so subtract one token, the subtraction operation would be combined with the "check if tokens remaining field is positive" operation. The value to be subtracted would be one in this case. The server site may also add tokens, effectively granting more usage privilege to a user, by using the operation "add value to the tokens remaining field" of the complex ACL.

3.3 Pay Per Use Policy

The pay per use policy enforces that a user has to pay for accessing data. For the SAC approach, the payment is always in tokens. It is the responsibility of the service provider to give the proper meaning to a token.

The implementation using the complex ACL is similar to the one described for the simple counter policy. The difference is that the server requests the cost of the data, expressed in number of tokens, to be subtracted from the tokens remaining field, instead of one token. For example, the server can specify a cost of three tokens with the "check and subtract tokens" with a comparison value of 3. This operation subtracts three tokens from the tokens remaining field and accesses the data only if the tokens remaining field is greater than three.

3.4 Temporal Counter Policy

The temporal counter policy is responsible for enforcing the restrictions that a user may access data that he/she is cleared for only a certain number of times in a certain period of time. For example, a user may be restricted to access a data item only 10 times per day.

This policy can be implemented by using the tokens remaining field of a complex ACL entry to represent the number of times a user may still access data. The reference field represents the time of last access, and the server uses its date value to extract a comparison time. The server uses two operations to enforce the policy. The server resets the tokens remaining field if the reference field in the client SAC is not the same as the comparison

time generated by the server using the "check and reset" operation defined in Table 3. Then, for each access the server needs to check if the tokens remaining field is greater than zero. If it is, one token is subtracted from the tokens remaining field and the access will be granted.

3.5 Simple Temporal Policy

The simple temporal policy is responsible for enforcing that a user may access data only for a certain period of time. This can be used, for example, for subscription services where a user has to pay monthly or yearly to access the service. This can also be used to allow a user to try a service for a certain time before having to pay for it.

This policy can be implemented by using a complex ACL's reference field as the day, month, year, or any other temporal parameter. The security attributes of the data being accessed specifies, using one of the "check if comparison value is the (same as, greater than, or lower than) the reference field" operations described in Table 3. The client SAC must check the complex ACL entry specific to the service, comparing the reference field with a value, which can represent the current day, month, year, or a temporal parameter. The access is only granted if the value satisfies the policy specification. For example, in order to allow access to data only in a specific month, the server could use the comparison value 7 to specify that the current month is July.

3.6 Temporary Limited Use Policy

The temporary limited use policy enforces that a user may access a data item only a certain number of times. The difference between this policy and the simple counter policy is that this policy is used for temporary accesses and assumes that the server adds the permission to access data only on the first time a user requests that data. The simple counter policy assumes that the server adds the permission independently of whether the user will ever access the data and service to which that permission refers. The temporary limited use policy could be used in a University, for example, to enforce that a student can buy only a certain number of tickets for a specific event.

The policy is implemented by adding an entry in the complex ACL with the respective use limit in the tokens remaining field the first time a user requests the service. Further requests for the service are checked by using the operation that allows access only if the tokens remaining field is positive, and if so, subtracts one token. This enforces the policy that the data is accessed only a certain number of times. However, the policy is used for temporary data and this implementation does not use the complex ACL efficiently since all past permissions would be stored in the complex ACL until the server SAC explicitly requests their exclusion. In addition, the service provider would have to keep generating unique access id's, e.g. event id's in the ticket example, until the explicit deletion of id's in all possible client SACs. A better implementation uses the reference field to store the expiration date of an access clearance. This implementation uses the "entry expiration by references" update operation, when a user connects to the server. The server uses its current date as a parameter to expire the entries. After that the server SAC requests a specific entry to be added to the client SAC complex ACL using the "add a label" operation. The reference field of this clearance represents its expiration data and the tokens remaining field represents the number of times the user can access the data. The "add a label" operation does not have any effect if that label is already present. This prevents the resetting of non-expired access labels. The user can then request access to the data, which has the security attributes specifying the number of tokens that must be subtracted.

As an example, assume that the current date can be represented by the number 21, the data attribute is 14, and the data can be accessed three times. The server would first expire all entries on the complex ACL with label 14 and expiration date (stored in the reference field) lower than 21. After that the server would add an entry allowing three accesses, and with an expiration date (e.g. expiration date that can be represented by the number 61). The data would then specify the "check and subtract tokens" operation with a comparison value of 1.

3.7 Good Before Policy

The good before policy enforces that a user is cleared to access some data a specific number of times, but only before a deadline. This can be used to allow a user to try a service up to a specific date, but not to try to access a huge amount of data, which might compromise the service. For example, a library may allow a user to try its service for a month but does not want the user to download all the data that the library has during this initial month.

This policy can be implemented by using the reference field of a complex ACL entry to represent the expiration time and the tokens remaining field to represent the number of times the user is allowed to access data. The server SAC requests the "token expiration by reference" update operation for all users that connect to this server SAC. It uses the server's current date as the value to compare to the reference field from the client SAC complex ACL. The data has security parameters specifying that a number of tokens must be deducted. The data will be accessed up to the number of times specified on the tokens remaining field of the complex ACL entry. The tokens remaining field will be zeroed after the expiration date stored on the reference field.

As an example, assume that the current date can be represented by the number 61, the data attribute is 98. The server would first zero all tokens on the complex ACL with label 98 and expiration date (stored in the reference field) lower than 61. The data would specify the "check and subtract tokens" operation with a comparison value that represents the number of tokens that must be deducted.

4 Online Banking Example

As a proof of concept for the Safe Area of Computation approach, two test bed systems were built that use the SAC approach for Internet transactions. These test beds use smart cards to implement client Safe Areas of Computation based on the Siemens SLE66CX160S integrated circuit and the CardOS M3.0 operating system. The functionality of the client SAC is implemented by applications loaded on the EEPROM memory.

The Siemens SLE66CX160S integrated circuit has 16 Kbytes of EEPROM (CardOS uses 116 bytes of EEPROM and an additional small amount to keep a file system, which will not be discussed in this paper). The applications that implement the client SAC functionality use 4,925 bytes. From the remaining 11,459 bytes, 10 bytes are used for the pin code and user's id (assuming a four byte pin code and four byte user id), and 141 bytes are used for the client SAC private key. Thus, there are 11,308 bytes that can be used to support service providers. Each service provider requires 141 bytes to store its public key plus the storage necessary for the ACLs. Therefore, the smart card used for the test bed can support 77 service providers that have security policies that require a maximum of four different simple labels (labels that can be accommodated in a simple ACL). Alternatively, it can support 30 service providers that implement security policies requiring all three ACLs with each having 32 entries. The smart card can, therefore, support a sufficiently large number of service providers, even if they require complex security policies.

The current test bed implementations impose some particular requirements for the client and server computers. The client platform requirements are:

a) Windows 98 or NT system;
b) A smart card reader;
c) A standard browser that supports ActiveX controls; e.g., Microsoft Internet Explorer or Netscape Navigator;
d) A set of programs that may be downloaded as an ActiveX package, which consist of the necessary controls and a program used for communication between the client SAC and the controls.

The server platform requirements are:

a) The server platform must be tamper resistant. This is a very strong assumption and as such requires very specific servers;
b) Windows NT server;
c) Microsoft Internet Information Service (MIIS);
d) A particular ISAPI extension that communicates with the server SAC;
e) The server SAC application;
f) A secure connection to the database that stores classified data.

These requirements can be easily met by a Windows platform, which is the platform used for the implementation. In addition, the server could be easily migrated to a Unix platform.

The first test bed was built for the Alexandria Digital Library (see [FRE98] for details about the Alexandria Digital Library) where it is used to download files to authorized persons in a secure way. This system is very simple and demonstrates only a few of the approach's capabilities. Because of this a second system was built to demonstrate how the SAC approach can be used to protect transactions for online banking. The details of the online banking system are described in the following subsections.

A generic online banking test bed was implemented to better illustrate some characteristics of the SAC approach, which were not demonstrated in the ADL test bed. This test bed implements a fictitious bank, called Bells Largo, which provides its clients with the ability of checking their account balance and/or performing transfers. The bank allows some users to only check balances, others to only perform transfers and still others to perform both operations. In addition the users can place some personalized restrictions on the transfers, such as limiting the amount per day.

Bells Largo distributes client SACs (smart cards) to all of its client. The clients are only able to access the online banking operations using properly equipped computers. This means that these computers must have card readers and the necessary programs installed.

Each client SAC is personalized to the needs of each client of the bank, allowing the client to check his/her balance and/or perform transfers. In addition, for each client, a restriction on the value that may be transferred per day is imposed.

The next subsections describe the ActiveX controls used for allowing users to interact with the bank in a safe and controlled way and the steps of a user interaction.

4.1 Authentication control

Every HTML page that uses the client SAC and can be bookmarked must have an embedded authentication control. The client SAC establishes a session key using the authentication process. Thus, a session key is not

established without authentication through an authentication control, and without authentication the client and server SAC cannot communicate.

The authentication control has 3 properties, 2 events and 1 method. The properties are used to provide the control static information (within that page), and the events and method are used to enable communication among controls.

The authentication control properties are *First Pointer*, *Service Provider*, and *Web Server*, and they are used as follows:

a) *First Pointer*. This property is optional. It is used if an authentication control wants to use its window to show pointers inside a container after authentication. The property was designed to support a convenient flow for users that access systems exclusively through browsers. It can provide a login page that shows options after a correct authentication. When used, this property is a string that starts with six digits and is followed by a pointer to a container. The six digits represent the security type, security label and comparison value of the data.

b) *Service Provider*. This is a required property and represents the service provider ID. The client SAC may support several service providers. It is this property, which is represented as a short integer that defines the service provider ID to which the user will be authenticated.

c) *Web Server*. This is also a required property. It represents the address of the web server that intermediates the communication with the server SAC. Although one might think this could be used for man-in-the-middle attacks, this is not the case. The SAC approach is protected against man-in-the-middle attacks without making any assumptions as to how the web servers are implemented.

The control events are *ChangePage* and *Authenticated*. They work as follows:

a) *ChangePage*. This event is designed to enable the authentication control to substitute for the page being shown to the user. It takes as a parameter the URL of the page that the control wants to show. This page will replace the page currently being shown to the user.

b) *Authenticated*. This event is used to signal that the client SAC has authenticated the user and itself to the server to any other control embedded in the same HTML page. Thus, it is used to synchronize requests for data and containers after authentication. The authentication control always broadcasts an *Authenticated* event after a successful authentication.

The control method is called *IsAuthenticated(void)*. This method is called by other controls to verify that the user has been authenticated. The authentication control responds to an invocation of this method by broadcasting an *Authenticated* event to all controls embedded in the same page, if the user has been authenticated. This enables a control to query the authentication control when it believes it missed the first *Authenticated* broadcast. This may, for example, occur due to initialization of the control.

The authentication control has a transparent background and uses a state machine to show messages to the user. If the user has not already been authenticated, the control requests a pin code from the user, as shown in Figure 2. In addition, the control requests the user to insert a card if a card has not yet been inserted in the card reader. After a card has been inserted in the reader and the user enters the pin code and clicks the ok button, the control shows a changing message while waiting for the SAC to authenticate all parties (user, client SAC and server SAC). The control shows different color boxes that say "authenticating" in each time, following the succession shown. This results in an animation effect, with the letters decreasing in size in a cyclic manner.

After the user has been authenticated the authentication control has different behaviors, depending on the presence of a valid *FirstPointer* property. The authentication control does not show anything, becoming transparent, if the *First Pointer* property is not present. The authentication control always fires an *Authenticated* event after a successful authentication.

Figure 2: Interface used by the authentication control to request pin codes

This control can be used in many different applications without change and is an important control that does most of the necessary initializations on the client side of the interaction.

In the online banking test bed the *First Pointer* property points to a valid container. There is a request for the container, after the user has successfully authenticated to the client SAC and the client and server SAC have mutually authenticated each other. After the container is properly requested from the server SAC, it is received by the Client SAC. The client SAC then hands the user the data item pointers for which the user is cleared in clear text representation. The authentication control represents these headers as buttons for the user's selection, as shown in Figure 3. The control does not make any assumption about the number of headers that must be shown as buttons. It just centers and equally spaces them. The page designer must make sure that a large enough area will be

reserved for the control to show all possible headers as buttons; otherwise, some of them may not be displayed, even though the user may want access to the data they represent. The control fires the *ChangePage* event after selection of one of these buttons. This continues the interactions through a new dedicated page. All information, such as the text to be shown in the buttons and the page to be changed to, are encoded in the description field of the headers.

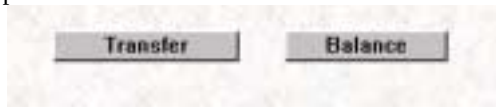

Figure 3: Control showing a container with pointers to two authorized operations

4.2 Download Control

The download control is a transparent control that does not show any information in its client window. This control is used to determine the pointer and security attributes of the restricted data that is supposed to be downloaded. The control uses pop up windows to allow a user to define a file name under which the data will be stored on the local computer. The control uses a standard Windows common file dialog box to allow the user to choose a name to save the downloaded file. In addition it uses the pop up window to inform the user about the progress of the download.

The download control has one property, one event and one method. The property is called *First Pointer* and is similar to the property from the authentication control with the same name. The difference is that this property points to data on the server and must always be present. It is this property that defines what data will be downloaded from the server.

The method of this control is called *OkAuthenticate(void)*. It is used to communicate to this control that the user has been authenticated. The request for the download of the data is performed only after this method is invoked.

The event is called *ReqAuthenticated* and is fired after the control has been initialized. This event invokes the *IsAuthenticated(void)* method of an authentication control to handle cases when its *OkAuthenticate(void)* method is called before proper initialization and the authentication acknowledgement is missed.

4.3 Rich Text Control

This control is used to show a text message to the user. The message is formatted in rich text format (RTF), which enables it to be displayed using the many features this format is able to specify, e.g. font, size, and appearance. This control is used to display restricted messages to a user. For example, the control that is used to show a user's balance is shown in Figure 4.

This control, like the download control, has one property, one method and one event. The property, method and event have the same functionality and behavior as the ones described for the download control. However, unlike the download control, this control is visible in order to show the text to the user.

Figure 4: Rich text control showing a formatted balance

4.4 IO Control

The rich text control is a very simple and efficient way to show sensitive text to a user. Sometimes, however, input from the user is needed. If this input is not sensitive, it can be done outside the control of the SACs. On the other hand, the SACs must be invoked when the input is sensitive. Because of this the IO control was developed.

The IO control is used to show sensitive text and receive sensitive input from the user. Its functionality is not a super set of the rich text control, since it cannot show text formatted as rich text. This may change in the future and the IO control may become the only control used for the output of sensitive text and/or input of sensitive data.

The IO control can show sensitive text, which is located on the top of the control. The sensitive text is followed by a certain number of input fields, window text boxes, that are used to input sensitive data. Each one of these text boxes may be preceded by text with a description of what input should be entered in that box. Two buttons are drawn in the bottom of the control so the user can confirm that all inputs have been entered or can cancel the operation. The IO control used for specifying the account to transfer to and the amount in the online banking system is shown in Figure 5.

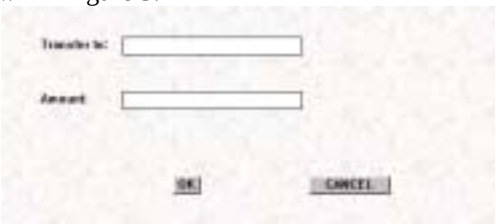

Figure 5: IO control with two input fields used for online transfers

Some or all of the inputs may be numbers that must be cleared by the complex ACL in the client SAC. Because each access control list entry has limited space, it is not

possible to clear large values. Therefore, the control may use a scale factor for the entered value. This scaling operation is used for translating a value to a token; e.g., every $ 20.00 is one token. The scaling value is embedded in the security attributes associated with the control and may truncate the value requested by the user. The SAC server will scale the value back up, before sending it for processing. This scheme is used in the test bed for demonstration purposes only. A real application would need to change the scale factor to give more flexibility to the meaning of the tokens, and in some cases the complex ACL may have to be expanded to have more than one byte in some of its fields.

The button in the bottom of the control is used to signal the control that all input has been entered and to keep the current state of the control. This is required in cases where the control changes its state as inputs are entered and sent to the server. An example of such an interaction will be shown for the transfer operation in the Bells Largo test bed.

The IO control has one property, one event and one method. The method and event are used in the same way as those used in the download control. The property is called *Pointer* and it is initially used for the same purpose as the *FirstPointer* property of the RTF and Download controls. The difference here is that this property changes each time a user submits input and it keeps track of the characteristics (security attributes and pointer) of the data being shown in the control.

4.5 User Interaction

A user will start interacting with the SAC when he/she chooses to perform an online banking operation. A web page with an authentication control is displayed to the user at the first interaction with the online bank that requires restricted access.

The user has to enter his/her pin code in order to authenticate him/herself to the client SAC. The authentication control shows a message requesting the user to insert the smart card (client SAC) in case he/she has not yet done so.

The test bed bank provides two services: balance and transfer. The authentication control has a pointer to the unclassified container that has these two classified data pointers. After the user enters a correct pin code the client SAC and server SAC mutually authenticate themselves and create a secure channel between them. After that, the authentication control requests the unclassified container, using the *FirstPointer* property that points to the container. The client SAC receives this request and validates it. If the validation is successful it requests the container from the server SAC. The server SAC fetches the container and sends it to the client SAC, together with the security attributes of the data pointers. The two data pointers have different classifications and the client SAC will only send the pointer(s) for which the user is cleared to the authentication control.

A user that has both clearances can choose to check his/her balance or to perform a transfer. The user clicks on the corresponding button in order to accomplish this. The authentication control fires its *ChangePage* event as a result of the button click. This event is used, together with a Javascript script to change the page shown in the browser to the appropriate one. The balance check and transfer operations are discussed in detail in the next subsections.

4.5.1 Balance Check

The balance check is a very simple operation in which sensitive text is presented to a user. The rich text control presents this data as nicely formatted text. It is this control, embedded in a web page, that is used for showing the balance.

A user may also request a balance page through a direct link, e.g. a bookmark. For this reason, the page has an authentication control embedded in it. This authentication control becomes transparent after the user's authentication.

The rich text control has a property called *FirstPointer* to indicate the security attributes of the data to be shown in the control, and a pointer to locate it. A user that sees this page after going through the main page will have been cleared to have this data; otherwise, he/she would not have received a button that would take him/her to this page. However, the SAC does not keep track of previous steps and does not take this into consideration. It only checks the *FirstPointer* property to decide whether or not the user can receive the data. A user that does not have the necessary clearance will not receive the data, being blocked either in the client SAC or in the server SAC. The blocking on the server SAC occurs in situations where the user tries to cheat the SAC and changes the security attributes of the *FirstPointer* property locally.

4.5.2 Transfer

The transfer operation is more complex and has many steps. The operation can also enforce many different policies. As an example, the test bed bank enforces the policy that: "some users may transfer up to X dollars per day". It uses the temporal counter policy to enforce this. For this particular case, an entry in the complex ACL has the fields:

- Security Label: This field represents the clearance that the user has to have to perform a transfer.
- Reference: This field represents the day of the last transaction.

- Tokens remaining: This field represents the amount that a user can still transfer on the reference day stored in the reference field.
- Reset Value: This field has the maximum value that may be transferred in a day (the X value in the policy). It is used to reset the tokens remaining field when the reference day is different from the current day.

Each step of the transfer operation depends on the previous one. Because of this these steps cannot be implemented as different web pages that can be randomly accessed. Therefore, the web page that allows a user to perform a transfer has an embedded IO control that keeps track of what step a user is performing.

The web page used for transfers is only shown to users that have the necessary clearance to perform a transfer. This is restricted data that the server returns to the client SAC; therefore, it has security attributes associated with it. Besides checking that the user is cleared to access this data, the security attributes have the function of performing an operation to test a value against the reference field of a complex ACL entry and to reset the tokens remaining field, if necessary. The value that the reference field is tested against is a value sent by the server SAC, which represents the current day. This will ensure that the tokens remaining field is always, and only, reset when the day of the last transfer is before the current day (as perceived by the server). If the current day is after the day of the last transfer, then the value sent by the server will be placed in the reference field of the complex ACL entry, and the value in the reset value field will be copied to the tokens remaining field.

The value entered will be part of the security checks to decide if the user can perform the operation. The account to which the transfer will occur will be part of the data request.

If the user clicks the "Cancel" button, the IO control sends a message to the client SAC requesting it to report this to the server SAC. If the user clicks the "OK" button the IO control composes the security attribute using the value of the transfer and the data pointer using the account number to which the transfer is to be made. The IO control fills a structure with these security attributes and the pointer from the FirstPointer IO control property. This structure is sent to the client SAC, which checks if the operation is allowed. The client SAC uses the complex ACL for this check and will only send the request to the server SAC if the user is allowed to perform the transfer for the indicated amount. To perform the check the client SAC subtracts the amount requested from the tokens remaining and uses this result to make its decision. It does not commit the result to the tokens remaining field at this time.

When the server SAC receives the request for the transfer it checks if the security attributes are valid, and requests the transfer. This still does not guarantee that the transfer will take place. The server SAC needs to receive a confirmation from the server database that the operation is allowed, e.g. the user may not have the necessary amount in his/her account. After the server SAC receives a confirmation that the transaction can take place, it sends a notice saying what was requested to the client SAC. The security attributes of this confirmation request that the transferred value be subtracted from the tokens remaining field of the client SAC complex ACL and committed. Thus, at this time the tokens remaining field stores a new value, which is the old value minus the amount of tokens that represent the transferred value. The client SAC then hands the data sent by the server to the IO control. This data specifies the next step using a pointer that will be used in the next request from the client to the server. This pointer will be stored in the *FirstPointer* IO control property. The data also specifies that the IO control should not draw any input field. This confirmation presents two buttons to the user: one that confirms the transfer and another that cancels it.

The ActiveX control shows the confirmation message to the user and waits for an interaction. If the user clicks on the "OK" button, the IO control sends the confirmation to the client SAC. The client SAC sends this confirmation to the server SAC using the secure channel. The server will only validate the transfer request when it receives this confirmation. The server then sends data that specifies a message of thanks for using the service.

If the user clicks the "Cancel" button, the IO control sends the cancellation to the client SAC. The client SAC sends the cancel request to the server SAC. After performing the cancellation, the server SAC sends a cancel confirmation to the client SAC. The IO control shows a confirmation of the cancel. This cancel confirmation has security attributes that request that the transfer value must be added back into the tokens remaining field of the client SAC's complex ACL. This will guarantee that canceled transfers do not penalize the user.

5 Conclusions and Future Work

The Safe Areas of Computation approach is very flexible, and it can accommodate many different security requirements. This paper described the three types of access control lists implemented on the client SAC. Because these access control lists are generic and provide flexible operations, the approach can be used to implement many different security policies. The basic security policies presented in this paper can be composed into more complex policies to implement most of the realistic policies currently in use.

Employing a user- or device-specific access control list enables the personalization of the accesses and restrictions a user will have (e.g., only allowing the user to view his/her bank account balance and not allowing withdrawals or transfers). This personalization has the advantage of providing access to only the services that the user signs up for, which also limits the user's liability, since the impersonation of the user is limited to only these services.

The test beds, which were used for safe Internet transactions, showed how the SAC approach can be easily implemented using ActiveX control technology. An advantage of the SAC approach is that standard applications only need to be enhanced to use the generic functions provided by the SAC to get secure access to sensitive data. A service provider that needs the security of the SAC approach will likely already have a security policy that associates security labels with its data. The service provider then only needs to design ActiveX controls that are in agreement with this policy and to provide a user interface. In order to make it even easier to implement new services using the SAC approach, this paper described some ActiveX controls that can be used to generically support new services. In addition, an area for future work is the development of tools for easing the implementation of Web pages using generic ActiveX controls. These tools would include a graphical user interface that provides an easy way for an administrator to interact with the server SAC. These interactions include adding new users, setting the number of times a user can access a specific data item, invalidating client SACs, and changing a user's access.

The test beds that were implemented demonstrated that the client side application need not require interactions from the user, other than what is necessary to manipulate data. In fact, the only time a user is aware of the SAC is when inserting the smart card in the smart card reader, when entering the pin code, and when trying to perform non-authorized operations. All the other interactions are carried out by the ActiveX controls and are transparent to the user. This results in an application that is user-friendly and allows the client and server SAC to enforce a security policy without requiring complex password schemes.

To provide different services, ActiveX controls must be incorporated into web pages. Almost all the time necessary for the design of new services that use the SAC approach is spent incorporating the correct ActiveX controls to enforce a particular security policy. The test bed implementations showed that it is highly probable that a set of basic ActiveX controls can be built to allow service providers to use the SAC approach for secure transactions through the Internet. An area for future research is identifying these controls, studying the feasibility of building them, and determining how well they fulfill different requirements. In addition, the design and development of a toolset that uses these ActiveX controls to aid in the construction of web pages for services that use the SAC approach must be further researched.

6 Acknowledgements

This research was partially supported by PulsePoint Communications and the University of California through a Micro Grant. It was also partially supported by Siemens AG, Mondex International, and Xerox Corporation.

References

[DDK97] F. De Paoli, A. L. M. dos Santos, and R. A. Kemmerer, "Web Browsers and Security", *Mobile Agents and Security*, LNCS vol. 1419, pp. 235-256, Springer-Verlag, 1998.

[DFW96] D. Dean, E. W. Felten, and D. S. Wallach, "Java Security: From HotJava to Netscape and Beyond", *Proceedings of 1996 IEEE Symposium on Security and Privacy*, pp. 190-200,1996.

[DK99], A. L. M. dos Santos and R. A. Kemmerer, "Safe Areas of Computation for Secure Computing with Insecure Applications", *Proceedings of the Fifteenth Annual Computer Security Applications Conference*, pp. 35-44, 1999.

[FRE98] J. Frew, et. al., "The Alexandria Digital Library Architecture," *Proceedings of the Second European Conference on Research and Advance Technology for Digital Libraries*, pp. 61-73, 1998.

[HOP96] D. Hopwood, "Java Security Bug (Applets Can Load Native Methods)," Risks Forum, vol. 17, no. 83, 1996, ftp://ftp.sri.com/risks/17/risks-17.83.

[LAND84] C. E. Landwehr, C. L. Heitmeyer, and J. McLean, "A Security Model for Military Message Systems," *ACM Transactions on Computer Systems*, vol. 9, no. 3, pp. 198-222, 1984.

[NIS99] NIST FIBS PUB 46-3, " Data Encryption Standard (DES)", National Institute of Standards and Technology, 1999, http://csrc.nist.gov/fips/fips46-3.pdf.