A Unified Security Framework for Networked Applications

Joerg Abendroth[*]
Distributed Systems Group
Trinity College, Dublin
Joerg.Abendroth@cs.tcd.ie

Christian D. Jensen[†]
Informatics & Mathematical Modelling
Technical University of Denmark
Christian.Jensen@imm.dtu.dk

ABSTRACT

Various security models have been proposed for different types of applications and numerous types of execution environments. These models are typically reinforced by adding code to the application, which authenticates principals, authorises operations and establishes secure communication among distributed software components (e.g., clients and servers). This code is often application and context-specific, which makes it difficult to integrate an application with other each other.

In this paper we propose a new unified access control mechanism that supports most of the existing security models and offers a number of additional controls that are not normally provided by security mechanisms. Moreover, the proposed mechanism integrates well with existing programming paradigms for distributed application, e.g., client/server technology and component based programming. This means that it can be seamlessly integrated with most existing distributed applications. We have implemented the proposed mechanism in a framework, that can be instantiated to implement different security models and policies. We present a qualitative evaluation that demonstrates the framework's ability to support a wide range of security policies and a preliminary performance evaluation of the framework.

Keywords

access control, active software capabilities, policy models

1. INTRODUCTION

A large number of different security models are proposed in the reviewed literature [1, 10, 6, 3, 22].

These models are generally formulated to reflect the requirements of a particular application domain. Mandatory access control models are used by the military, while discretionary access control models are often considered adequate for academic institutions and home computers. A mixture of mandatory models (e.g., role-based access control) and discretionary models are often used in industry.

It is often possible to support different security models with the same basic security mechanisms. However not all models can be accommodated by all security mechanisms, e.g., it is impossible to support a mandatory access control model with a simple capability-based access control mechanism [13, 8]. In many cases, the security model is designed with great care, but the supporting access control mechanism and security infrastructure are simply designed to satisfy the requirement of the particular model. This means that there has been little effort to design a flexible security mechanism that can support multiple security models. Typically, minimalism of mechanism is the only approach taken to provide a flexible infrastructure and providing sufficient support for complex security models is left as an exercise to the application programmer. However, developing powerful security mechanisms is a highly specialised task that should be performed by adequately skilled developers with the support of good development tools.

Recent developments in software engineering, such as the integration of legacy applications and component-based programming, often require the re-engineering of large applications or components that may have been developed with different security models in mind. This introduces the problem of mapping security events from one model to another, e.g., a client that relies on Kerberos tickets [14] may have to interface with a server that implements role-based access control (RBAC) through X.509 [20] attribute certificates. Finally, the web has become the principal mediator of business information, and companies are increasingly providing restricted access to internal business data and processes, typically through web-portals [26]. The web portal must provide clients with a simple and flexible security mechanism that can mediate access to legacy applications that implement a variety of different security models developed for different application domains.

In this paper we propose a new unified security mechanism, called *active software capabilities*, which combines the ideas of *proxy-based security* [19, 17, 9] and *active capabilities* [28] to achieve adaptability without the need to rewrite application code. Security proxies are downloaded at runtime from a security server, which is associated with the *object server* that manages the resource that the client wishes to access. The security server can either be an ex-

[*] This work is sponsored by a research grant from IONA Technologies PLC.

[†] This work was completed while the author was working at Trinity College, Dublin.

SAC 2003, Melbourne, Florida, USA

ternal server, like the ticket-granting server in Kerberos, or it can be co-located/integrated with the object server. The *client proxy* establishes a secure communication channel with a *server proxy* that runs on the object server. The client proxy uses this channel to forward the necessary credentials to the server proxy. These credentials can be simple data structures, e.g., usernames or certificates, but they can also be encoded in an executable *policy object*, which corresponds to an active capability [29, 28]. The policy object is signed by the security server and evaluated by the server proxy. Downloading client and server proxies at runtime allows transparent mapping of security events and credentials from one security model to another, e.g., an identity certificate can be transformed into a role membership certificate with the help of an external RBAC server and without modifying the code of the client and the server. The ability to encode credentials as executable content facilitates the development of flexible security policies, i.e., the definition of a security policy is limited only by the expressive power of the scripting language used to encode the policy object.

We have implemented the active software capability mechanism and a number of different security policies in a framework written in Java. The framework uses an unmodified version of the lookup server provided by JINI [18, 30] to implement the external security server. This means that applications written for JINI can use our framework without modification. Moreover, the flexibility and adaptability of the proposed mechanism allow it to interface with existing security protocols and security infrastructures, e.g., to use existing public key infrastructures (PKI), which means that the framework can be seamlessly integrated in most organisations.

The framework has been evaluated by implementing a number of different security proxies that enforce different security policies without requiring modifications to the client application. These policies demonstrate that the framework can be instantiated to enforce a large number of different security policies. We have also performed a preliminary performance evaluation, which indicates that the overhead introduced by the mechanism is relatively small and that the division of responsibilities between client proxy and server proxy may be used to offload work from the server to the client, thus improving the scalability of the system.

The rest of this paper is structured as follows: Section 2 presents related work in the areas of proxy-based security and active capabilities. The design of the unified security framework is presented in Section 3. Section 4 describes our implementation of the framework, Section 5 presents our evaluation of the framework and Section 6 describes our conclusions and directions for future work.

2. RELATED WORK

The active software capability framework combines ideas from proxy-based security and active software capabilities. Related work in each of these areas is presented in the following.

2.1 Active Capabilities

The active capability model was developed at the University of Illinois at Urbana-Champaign [29, 28]. It is an extension to the traditional capability model [7, 15], which replaces a fixed set of permissions with a script that is evaluated by the server in order to determine the access rights of the caller. This allows additional controls to be imposed at runtime, such as limiting the time of day when a resource can be accessed or only allowing access when the server system load is below a certain threshold. However, active software capabilities have to be managed explicitly by both client and server, and all calculations related to the access control decision is performed by the server. Active software capabilities extend the existing active capability model with a client proxy, which allows us to offload work from the server to the client, e.g., collecting and refreshing certificates, thus improving scalability of the system.

2.2 Proxy-Based Security

The active software capability model is inspired by work on hidden software capabilities in the SIRAC project at INRIA Rhône-Alpes [9, 12]. Hidden software capabilities separate security from the functional aspects of an application, by delegating management and exchange of capabilities to protection interfaces interposed between client and server. However, hidden software capabilities define a fixed set of permissions that cannot be modified without replacing the protection interfaces.

The functions of the client proxy are similar to the restricted proxies proposed by Neumann [19]. However, their work mainly focused on providing accountability in the security infrastructure, which we implement through administration policies. Another difference is that we do not require the client (restricted) proxy to be the only way to gain access. If a specific implementation wishes to handle the protocol correctly, without using the dedicated proxy and benefiting from the adaptability offered by active software capabilities, it is possible to bypass the client proxy and contact the object server. This demonstrates that the security depends on the policy objects and the certificates and not on particular pieces of code.

Related to our idea of using credentials to simplify the access control decision is the certificate-based access control system for widely distributed resources [27]. The Akenti framework distinguishes between three certificate types, which are used by the server for the access control decision. The active software capability framework uses credentials in the form of active objects, which means that our policy definition is more flexible.

Finally, previous work done on flexible management for security policies [31, 16] has been of great influence. However, their framework still requires that a specific infrastructure exists, i.e., a policy editor can distribute the evaluation tokens via the policy server. Independent management of two separate domains, e.g., two companies in a virtual enterprise, becomes impossible. A further advantage of our framework is the flexibility of the actual policy object implementation.

3. ACTIVE SOFTWARE CAPABILITIES FRAMEWORK

To describe the active software capabilities (ASCap) framework we first present the idea of policy objects and then introduce the abstract layer model to explain the design in detail. Thereafter a process overview is also given.

3.1 Idea of Policy Objects

The aim of the ASCap framework is flexibility. The central idea is to achieve the notion of policy objects, which are no longer embedded in the object server access control component, but is an object delivered by the client. A policy object is cryptographically protected and originate from a trusted source. The client has the freedom to chose the most convenient policy, while the object server has the control which subset of all policies are allowed for a certain request. In cases where great flexibility is required an object server might allow any policy of the trusted source, therefore enabling it to introduce new policies after installation of the framework. Only the ASCap proxy on the client side would need to acquire the new policy objects and send them to the object server.

3.2 Design and Process Overview

In order to define our framework, we have identified four layers, which depend on each other as shown in Figure 1.

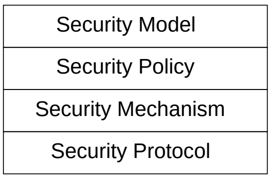


Figure 1: Unified security framework

The *security model* defines the underlying principles and the scope of the security policies.

The *security policy* specifies rules that must be enhanced by the access control mechanism layer.

The *security mechanism* combines the basic security mechanisms of the underlying system and the basic security infrastructure to enforce the security policies defined by the security policy layer.

The *security protocol* is the lowest layer available. Its main tasks are establishment of secure communication and authentication. The security protocol layer needs to be able to support the policies and mechanisms defined above.

The framework must provide a flexible and adaptable security mechanism that can support manifold security models and policies. Moreover, it should be able to employ different security protocols and interface with existing security infrastructures.

It is widely accepted that capability-based security mechanisms are more flexible than their ACL-based counterparts, so we have chosen to base our mechanism on capabilities [15]. This mechanism is described below.

Active Software Capabilities extend the active capabilities model [29, 28], where a small program, or script, is stored in the ASCap and executed by the server. The result of executing the active capability program indicates whether access should be granted or not. The ASCap model extends the basic active capability model with the notion of a proxy [24], which provides adaptability on the client side as well as on the server side. An overview of the ASCap framework is shown in Figure 2.

Before a client can access a server it must download an ASCap from an ASCap proxy provider, this can be either the actual server or a separate entity. Once downloaded, the ASCap can be stored on disk for later invocations of the same server (a timestamp and time of validity can be included in the ASCap in order to facilitate timely revocation). This step is shown in Figure 2–1. Both the client and server must instantiate their respective proxies, as shown in Figure 2–2. Figure 2–3 illustrates that the client ASCap proxy may request further credentials from external security services. These credentials could be identity certificates in an ACL-based model, a role membership certificate in an RBAC model or a proof of payment in an e-Commerce application. Finally, as shown in Figure 2–4, the client invokes the server through the ASCap proxy. The ASCap proxy adds a policy object and any additional credentials to the operation and parameters that are supplied by the client. Ultimately the server invokes the policy object, passing the operation, parameters and certificates to it. The policy object returns a simple yes/no answer, which is used to allow or disallow access to the server.

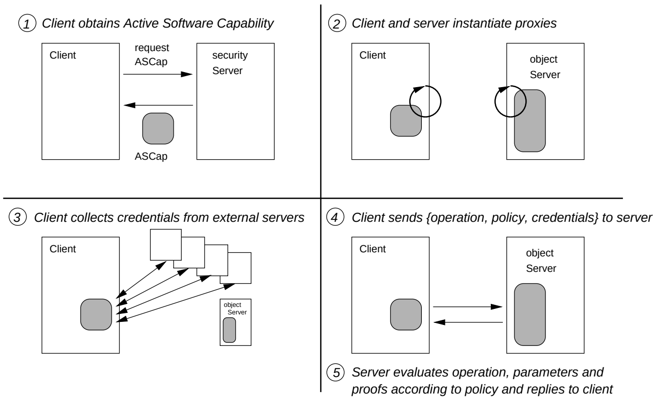

1 Client obtains Active Software Capability
Client
request
ASCap
security
Server
ASCap
2 Client and server instantiate proxies
Client
object
Server
3 Client collects credentials from external servers
Client
object
Server
4 Client sends {operation, policy, credentials} to server
Client
object
Server
5 Server evaluates operation, parameters and proofs according to policy and replies to client

Figure 2: Overview of the ASCap framework

Security Model Layer

This is the most abstract security layer, and it has received a lot of interest from the research community. Security models such as mandatory security, discretionary or ACL-based models require certain properties or mechanisms from the lower layers. New security models may be developed which require different properties from the lower layers. The goal of the ASCap framework is to provide support a wide variety of different and compound security models. This is achieved by providing lower layers, which flexibility is embedded already in the design.

Security Policy Layer

In the ASCap framework, security policies are defined by policy objects, which may examine both the parameters and the credentials provided by the ASCap as well as external server variables, e.g., the time of day, system load, etc. The policy object is stored either directly in the ASCap, or the ASCap contains a reference that allows the server to access the policy object from a trusted repository. Credentials can be simple values, certificates or data of any formal structure.

The use of separate policy objects ensures the adaptability of our framework. In the simplest form, the server proxy will do all required verification, request credentials and evaluate the policy according to supplied parameters. In this setup the client needs to hand the ASCap all authentication tokens.

It is important to note that at the current state a policy object can take any form that can be programmed in Java. We are investigating different policy description languages in

OPERATION, POLICY_OBJECT, CREDENTIAL_1, . . . CREDENTIAL_N

Figure 3: Structure of an active software capability

order to identify appropriate abstractions for a policy specification language that can then be compiled into Java policy objects. However the main contribution of this framework is to provide flexible lower layers, namely the policy distribution mechanism.
To design our framework we have identified three classes of policy objects:

Simple Policies: Only parameters supplied by the client are considered by the policies, e.g. a username password combination.

Parametered Policies: Both client and server need to provide parameters, e.g., security domain name. The policy object will either compare the parameters or in another way decide if access should be granted, e.g. a user and password, which is related to a security domain - this means that non-interference [21, 4] policies should be possible, by having the server check whether a client of a higher security level is currently using the same server.

External Server Policies: One or more of the parameters do not derive directly from the server or client, but is requested from an external server. This might require additional authentication, which will be discussed later. Policies including a protected certificate repository like a Kerberos ticket granting server [25], role membership certificates, current network load or objective checks on the clients status and identity.

We would like to emphasise that the external server policies enable our security framework to instantiate different security models and architectures without modifying client or server. Simple policies can be used to implement a pure capability system. By requiring a role certificate an RBAC system can be formed. Combining simple policies with external server id certificates, it instantiates an identity-based capability system, which is able to support multilevel security policies [8]. Moreover, external server policies are not limited to one external server, which allows decentralised administration of the security framework.

Security Mechanism Layer

The ASCap proxy itself is a proxy object, which is sent by the object server to a security server upon service registration. This enables a proxy-based approach to security [19, 9, 2]. The client downloads the ASCap proxy from the security server and instantiates it. Prior to accessing the object server, the ASCap proxy will collect all necessary credentials. Only if it can acquire these will the access request, in the form of an ASCap, be sent to the server. This reduces the number of failed client invocations to the server. The structure of the ASCap is shown in Figure 3.

Generally, the ASCap will be signed by the ASCap proxy, which is done to ensure accountability. The policy and/or credential objects may be encrypted in order to prevent the client from learning the details of the policy.

Security Protocol Layer

A security protocol which is flexible and secure enough to provide a reliable transport layer for various setups is needed. We have chosen to extend a simple session setup protocol, which assumes that the public key of the object server is known to the ASCap proxy. A shared key secure channel is opened during the session initialisation. This is required if in later phases an auditing server is used and a session identification connected to the clients real id is necessary.

4. IMPLEMENTATION

The implementation of the ASCap framework defines one passive entity: *the active software capability*, and three active entities: *the client*, *the object server* and *external security servers*.

The ASCap proxy connects the client to the server and implements the application security model in the interfaces between program components, i.e. outside the code that implements the functional requirements of the application. The client and server are traditional client/server programs that implement the functional requirements of the application. Finally, the external security services are required to support different security policies within the same framework.

Client

A client application can be any application that supports the ASCap interface. No additional security code needs to be added to the application. The ASCap interface is currently based on JINI [30, 18], so most applications that use JINI automatically fulfil this requirement.

Invocations are made to the server through the ASCap proxy, which may request additional credentials from external security servers before the actual server is invoked.

Object Server

The server runs a default proxy that accepts ASCaps from the clients, extracts operations, parameters, policy objects and credentials from the ASCap. The proxy evaluates the policy object, passing all the other elements as parameters. If the policy object returns a positive indication, the server object is then called with the parameters provided in the ASCap. Once the policy object has been instantiated, all subsequent calls from the same client use that instance, so the overhead is reduced to a single method call.

The implementation of this caching mechanism uses a hash table. The policy name acts as the keyword, which allows the required credentials and allowed commands to be accessed.

The server grants access once it has verified that the credentials are correct and that the command is permitted by the policies.

External Security Servers

External Security Servers are, like all external servers, an additional component that can be added to create the desired security framework. They are not part of the functional requirements of the client, but enable other services to be used. For that purpose, the external servers can provide the clients with different credentials after performing their checks. For example, this can be used to implement an $RBAC_1$ system, by letting the client proxy request a role membership certificate from an external role server. The role hierarchy and evaluation of inherited membership can also be evaluated at the external role server.

5. EVALUATION

In this section, we first give a short reasoning about the security and demonstrate the flexibility of the framework by showing how to implement a pure capability system and how to reuse the setup to implement an ACL based system. We then give a detailed description of our implementation of a system, that implements $RBAC_2$ [22]. Finally we present our preliminary performance evaluation of the developed prototype. A security evaluation is currently prepared and would be too complex to include in this paper. The main point is, that the security in the ASCap framework relies on the correct handling of the session protocol. A valid ASCap will yield to access, but may depend on external security servers, which are part of the trusted computing base.

5.1 Discretionary Access Control Setups

These are the two simplest access control policies. The server needs to verify the provided certificates.

5.1.1 Simple Capability-Based Setup

We first demonstrate how a simple capability-based mechanism can be instantiated within the proposed framework. Possession of a capability is a necessary and sufficient requirement to grant access to a protected resource, which means that a simple policy is sufficient. We have chosen a cryptographically secured simple policy (cf.3.2 Security Policy Layer) where the capability is stored by the client and is valid for a limited amount of time. The corresponding system is shown by the black part in Figure 4. The client retrieves a cryptographical protected credential, sends it in the ASCap to the object server, which verifies the correctness via a PKI chain.

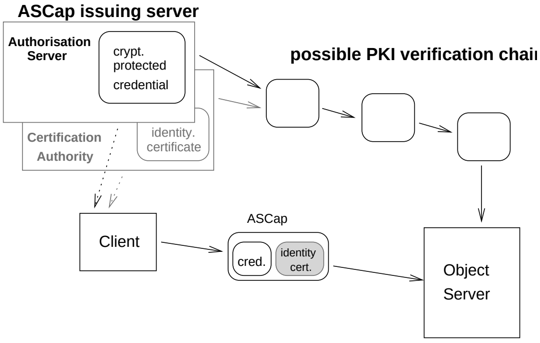


Figure 4: Simple capability-based setup with extension to an ACL-based setup in grey

5.1.2 Simple ACL-Based Setup

The capability-based setup can be extended into an ACL-based setup, where the client obtains an identity certificate from an external certification authority. The policy object contains a reference to the relevant ACL for the particular service. The ACL could be stored in the policy object itself, but that would require a short lifetime for the policy object in order to allow quick revocation of access rights. Since the policy object is included with the ASCap proxy, this means that the lifetime of the ASCap proxy must also be short. The extension to an ACL-based setup is shown by the grey part in Figure 4. The client collects an identity certificate and a cryptographical protected credential from the authorisation server. It sends both in the ASCap to the object server, which verifies it via the PKI chain.

5.2 Role-Based Access Control Setup

By changing the ASCap proxy to use external server policies the system behaves differently, but the application code does not need to be changed. Below is a detailed description of an $RBAC_2$ system, which implements a basic $RBAC$ system with additional constraints.

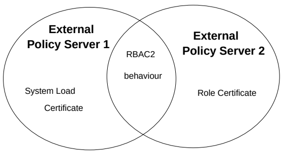
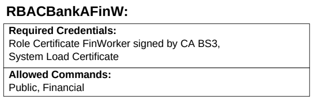


Figure 5: Overall system behaviour


RBACBankAFinW:

Required Credentials:
Role Certificate FinWorker signed by CA BS3,
System Load Certificate

Allowed Commands:
Public, Financial

Figure 6: Policy Object RBACBankAFinW

To instantiate this system the policy object would require two credentials from external servers. Each external policy server will influence the overall system behaviour. In Figure 5 credentials of two external servers are used, one with RBAC behaviour, and the other checking the system load. Together they result in an $RBAC_2$ behaviour.
The policy accordingly would look like that shown in Figure 6. The client fetches the two credentials and the server only has to verify the signature. The same behaviour would be achieved by sending an executable policy object, which queries the system load at the server. The difference occurs when the processing takes place.

The detailed access cycle is shown in Figure 7, which is described below:

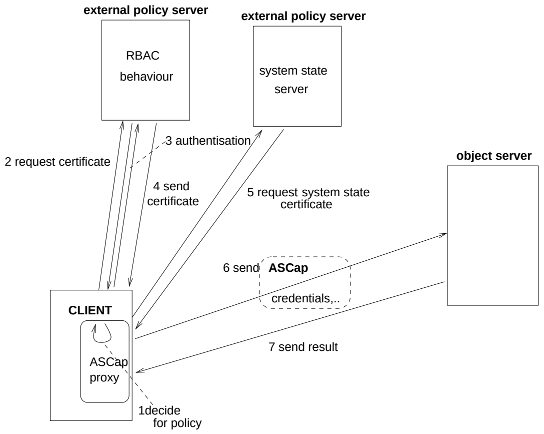


Figure 7: Access Request Cycle

Once the client issues an access request the ASCap proxy internally checks if the request is not denied by hard-coded rules. Then an appropriate policy is identified (Figure 7–1) and collection of credentials is started. The first (Figure 7–2) will be from an external policy server, which acts as the role server. The client must invoke a proper authentication with this server (Figure 7–3) and will receive a cryptographically secured certificate stating its role membership (Figure 7–4). Additionally a second external policy server must be queried, which issues a cryptographically secured system state certificate (e.g. system time or current load, Figure 7–5). This certificate needs to be short lived due to the nature of rapid system state changes. The ASCap proxy sends both certificates collectively inside the ASCap to the object server (Figure 7–6). At the object server, validity of the certificates is checked by verification of the PKI chain leading to the credentials. After executing the access request the result will be sent back to the client (Figure 7–7).

5.3 Performance Evaluation

We have measured the cost of instantiating the framework and initialising an active software capability in order to evaluate the overhead imposed by the framework.

The measurements of Figure 8 were made on a Pentium III 300, 128MB Ram, Linux 2.2.18, JVM 1.3.1-b24 mixed mode. The times are in milliseconds. We measured the time at the client application, and it corresponds to the time taken from the call to the ASCap proxy until the result of the access request is delivered back to the application. The 1st access column shows loading of the ASCap proxy relevant classes and including context switching. Subsequent accesses measure the real access request time required by the framework.

The first measurement (Null Invocation) compares the performance of the basic framework to SSL. In this setup no access control is done. After session initialisation, our framework is approximately 10% faster than a normal SSL connection. This is mainly attributed to the fact that we use AES [5] instead of Blowfish [23], which is used by SSL. Session initialisation primarily involves sampling randomness to be used by the cryptographic library.

In the further $RBAC_3$ setups with 1, 5 and 10 credentials are shown. This emulates security models, which take one or more external servers, where the behaviour of the external servers influence the overall security model. Finally, the same measurement is repeated for a $RBAC_2$ setup, where 1 credential and system load-checking on the server side is required.

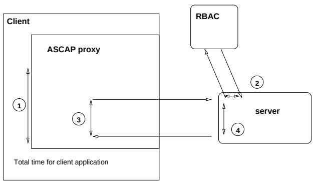


Figure 9: Traditional RBAC setup

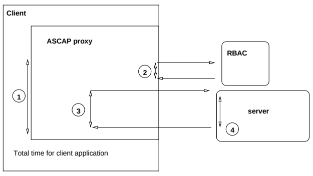


Figure 10: ASCap RBAC setup

The ASCap setup allows us to improve scalability by offloading work from the server to the client, e.g., the task of collecting credentials. We performed an experiment shown in Figures 9 & 10 to investigate the impact of shifting this workload from the server to the client. A traditional RBAC setup requires a form of database or storage of RBAC membership at the server. A client will only need to send its authenticated identity and the access control decision is performed entirely by the server. For comparison reasons we have stored the information in an external RBAC server. In our ASCap client-based setup, the client retrieves the role membership certificate, packs it into the ASCap object and sends it to the server. The server in this scenario will only verify the credentials.

The total time seen by the client application increases by more than 50% compared to the traditional setup. This is due to an unfortunate effect of Java serialisation/de--serialisation that we are currently working on eliminating. In a working environment, the client will cache the received role membership certificates and on each subsequent request gain 51ms. This means that the cost of retrieving, decoding and re-encoding the membership certificate is amortised if the client invokes the server 5 times with the same role membership certificate.

As expected, the RBAC server call takes the same time (36ms) regardless of who is doing it, while the work on the server increases by 40% (91ms simple verification to 164ms-36ms=128ms retrieval and verification) in the traditional setup compared to the ASCap setup. An additional advantage of the ASCap client-based solution is that at large-scale cases the workload and resource usage is distributed over the clients. Bottlenecks at the object server caused by resource constraints are avoided, e.g., a reduced number of network sockets (to clients and external security servers) are needed.

6. CONCLUSIONS AND FUTURE WORK

In this paper we examined the problems arising from integration of applications and software components that have not all been developed for the same security model. We identified the need for flexible security mechanisms and protocols that can accommodate the most commonly used security policies within a single framework.

We proposed a new security mechanism based on active software capabilities, and demonstrated how this mechanism is able to support a wide range of discretionary and mandatory access control policies. Security policies are enforced by verification on the object server, while the overall system semantics can be modified by external security servers and ASCap proxies that are instantiated at runtime independently of the application code. This allows us to construct a framework that can support multiple security policies for different objects at the same time by mapping principals and credentials from one security model to another, as described in the web-portal scenario.

We presented our design and implementation of the basic security mechanism and showed how the mechanism can be extended to support different security policies. We are currently developing a complete security framework for multi-policy security based on these extensions. We wish to integrate this framework with a web-portal service in order to verify that the framework facilitates mediation between clients and servers in different contexts. Having a fully functional and flexible multi-policy framework will allow us to proceed in further research in security policy interaction [11], as well as to do automatic proxy generation for high level policy specifications.

Finally, an important property of the ASCap framework is the notion of interface based protection programming. This means that specialised security professionals can define the security model and specify policy objects, which are then

Operation	1st access (ms)	subsequent accesses (ms)
Null invocation	1043	339
SSL	-	372
$RBAC_3$ 1 credential	1091	413
$RBAC_3$ 5 credentials	1230	522
$RBAC_3$ 10 credentials	1324	614
$RBAC_2$ (system load at server)	1419	560

Figure 8: Measurements of different ASCap setups

Scenario	1)Total time	2)RBAC call	3)client call	4) server processing
ASCap (Figure 10)	440	37	160	91
Traditional RBAC (Figure 9)	214	36	211	164

Figure 11: Times measurements for Figures 9 & 10

used by application programmers, thereby improving the overall security of the system.

7. REFERENCES

[1] D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations and model. Report MTR 2547 v2, MITRE, November 1973.

[2] M. Burnside, D. Clarke, T. Mills, D. Devadas, and R. Rivest. Proxy-based security protocols in networked mobile devices. In *Proceedings of the ACM Symposium on Applied Computing*, March 2002. *(to appear)*.

[3] D. Clark and D. Wilson. A comparison of commercial and military computer security policies. In *IEEE Symposium on Security and Privacy*, pages 184–194, 1987.

[4] C.O'Halloran. A calculus of information flow. In *ESORICS 90*, pages 147–159, 1990.

[5] J. Daemen and V. Rijmen. Aes proposal: Rijndael, 1998.

[6] D. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[7] R. S. Fabry. Capability–based adressing. *Communications of the ACM*, 17(7):403–412, July 1974.

[8] L. Gong. A secure identity-based capability system. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 56–63, 1989.

[9] D. Hagimont, J. Mossiere, X. R. de Pina, and F. Saunier. Hidden software capabilities. In *International Conference on Distributed Computing Systems*, pages 282–289, 1996.

[10] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[11] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In J. Peckham, editor, *SIGMOD International Conference on Management of Data*, pages 474–485. ACM Press, 1997.

[12] C. Jensen and D. Hagimont. Protection reconfiguration for reusable software. In *Second Euromicro Conference on Software Maintenance and Reengeneering*, pages 74–81, Florence, Italy, March 1998.

[13] P. A. Karger and A. J. Herbert. An augmented capability architecture to support lattice security and traceability of access. In *IEEE Symposium on Security and Privacy*, pages 2–12, 1984.

[14] J. Kohl and C. Neuman. The kerberos network authentication service (v5). RFC 1510, Digital Equipment Corporation/ISI, September 1993.

[15] H. M. Levy. *Capability-Based Computer Systems*. Digital Press, Bedford, Massachusetts, 1984.

[16] D. A. Marriott, M. S. Sloman, and N. Yialelis. Management policy service for distributed systems. Technical Report DoC 95/10, Imperial College, London, 1995.

[17] D. B. Marvin M.Theimer, David A.Nichols. Delegation through access control programs. In *12th International Conference on Distributed Computing Systems*, pages 529–536, 1992.

[18] S. Microsystems. Jini^{tm} architecture specification, version 1.2, December 2001.

[19] B. C. Neumann. Proxy-based authorisation and accounting for distributed systems. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 283–291, May 1993.

[20] T. S. S. of ITU. *Information Technology — Opens Systems Interconnection — The Directory: Authentication Framework X.509*. International Telecomunication Union, 1993. Standard international ISO/IEC 9594–8 : 1995 (E).

[21] Ryan and Schneider. Process algebra and non-interference. In *Proceedings of The 12th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.

[22] R. S. Sandhu, E. J. Coyne, and H. L. F. and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[23] B. Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). *Lecture Notes in Computer Science*, 809:191–204, 1994.

[24] M. Shapiro. Structure and encapsulation in distributed systems: The proxy principle. In *Proceedings of the 6th International Conference on Distributed Computer Systems*, pages 198–204, 1986.

[25] B. C. N. J. G. Steiner and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Winter 1988 USENIX Conference*, pages 191–201, Dallas, TX, 1988.

[26] I. Technologies. The iona iportal application server. Technical report, IONA Technologies, 1999.

[27] M. Thompson, S. M. W Johnston, G. Hoo, K. Jackson, and A. Essiari. Certificate-based access control for widely distributed resources. In *Proceedings of the Eighth USENIX Security Symposium*, pages 215–228, 1999.

[28] W. L. Tin Qian. Active capability: An application specific security and protection model. Technical report, University of Illinois at Urbana-Champaign, 1996.

[29] T. D. Tock. An extensible framework for authentication and delegation. Master's thesis, University of Illinois at Urbana-Champaign, 1994.

[30] H. Wong. *Developing Jini[TM] Applications Using Java 2[TM]*. Addison Wesley Longman, 2001.

[31] N. Yialelis and M. Sloman. A security framework supporting domain based access control in distributed systems. ISOC Symposium on Network and Distributed Systems Security, 1996.