Differential Data Protection for Dynamic Distributed Applications

Patrick Widener and Karsten Schwan
College of Computing
Georgia Institute of Technology
{pmw,schwan}@cc.gatech.edu

Fabián E. Bustamante
Department of Computer Science
Northwestern University
fabianb@cs.northwestern.edu

Abstract

We present a mechanism for providing differential data protection to publish/subscribe distributed systems, such as those used in peer-to-peer computing, grid environments, and others. This mechanism, termed "security overlays", incorporates credential-based communication channel creation, subscription and extension. We describe a conceptual model of publish/subscribe services that is made concrete by our mechanism. We also present an application, Active Video Streams, whose reimplementation using security overlays allows it to react to high-level security policies specified in XML without significant performance loss or the necessity for embedding policy-specific code into the application.

1 Introduction

Distributed applications and end users interact by dynamically sharing data, exchanging information, and using or controlling remote devices. In scientific endeavors, for instance, researchers remotely access resources like microscopes[4], 3D displays[28, 12], and may even wish to operate sophisticated components like the Tokomac fusion facility. In industry, companies share parts designs[10] or other data critical to their operation. Examples include Schlumberger's oil exploration processes where reservoir simulation data produced in computer centers should be shared with 'on site' personnel conducting drilling[32], and where simulations should use well logs to refine current drilling procedures. Another example is the airline industry, as with Delta Air Lines' sharing of flight and passenger information with third parties who distribute such data to select passengers for cellphone-based passenger notification[26]. Finally, in remote sensing and control, radar or camera data or telemetry/biometric information is captured, forwarded to, analyzed, and used by interested remote parties, sometimes involving remote control loops, as in telesurgery and targeting.

In many such applications, remote users are not interested in and/or should not see all of the data all of the time. Also, the criteria for these "which/whether" decisions can change rapidly. In fact, dynamic interest changes sometimes help make the implementation of such systems or applications feasible, by enabling dynamic data reduction[35], or they are used to optimize implementations, as with lossy multimedia[22]. Consequently, there are conceptual models for such changes, including context sensitivity[14] in human-centered ubiquitous applications, spatial or temporal locality in pervasive and distributed systems[36, 6], and current focus or viewpoint in remote sensing, graphics, and visualization[21]. Finally, whether implicitly determined or explicitly captured by quality of service expressions[29, 30, 3], the occurrence of dynamic interest changes in applications and systems is accompanied by the wide range of effects they can have, starting with simple changes in data selectivity applied to ongoing information exchanges[21], continuing with the need to apply varying transformations to data[22, 28, 24], and also including real-time control reactions as in dynamic sensor repositioning or in telepresence[9] or teleimmersive applications[31].

Security and Protection in Dynamic Data Systems. The general problem addressed in this paper is:

- How can appropriate security and protection can be associated with the data exchanges that take place in dynamic systems and applications?

In remote instrumentation and sensing, for instance, costly physical infrastructure must be protected from unauthorized or inappropriate access. In remote telemetry, privacy concerns may prevent us from implementing key safety functionality, as evidenced by applications like smart cars[20] or remote biometric monitoring. In cooperative scientific and engineering endeavors, end users wish to protect certain elements of the data being shared, such as the high resolution reservoir modeling data Schlumberger cannot make available to its competitors, or certain materials properties which parts designers do not want to disclose. Similarly, in remote monitoring and e-commerce, it is critical to ensure that only certain elements of data streams are made available to remote parties, as with airlines' caterers who should not receive data about passenger identities but must know about their food preferences, or as with the disclosure of passenger or tracking information to federal agencies in cases of potential criminal activities.

Differential Data Protection in Dynamic Data Systems. The target systems and applications addressed by our work are distributed applications in which continuous data streams are produced or captured, distributed, transformed, and filtered, in order to make appropriate data available where and when it is needed[28, 24]. The specific problem we address for such applications is that:

- developers typically organize the data being exchanged to meet functional needs, whereas

- security requirements may require different data organizations, distributed, and access patterns.

A simple example is a distributed sensor application in which data captured from multiple remote sensors is combined into a larger composite stream, as needed for sensor fusion or simply to take advantage of bandwidth improvements derived from the use of larger messages, for example. Programs operating on the composite stream can access all of the captured data, thereby increasing the potential damage from security violations. In this case, the problem to be solved is to protect the composite stream such that its data can only be accessed and used differentially.

Differential data protection for a data stream is defined as the ability to: (1) give only certain users or programs access to the data being transported or stored, (2) protect individual entries in data items, as when an airline provides caterers access to select portions of passenger records (e.g., indications of food preferences), and (3) limit the transformations and manipulations (i.e., services) that may be applied to data, as when preventing certain data manipulations that can extract or derive sensitive data (e.g., identifying faces in captured video). The data exchanges explored in detail in this paper derive from our ongoing research in sensor systems[2, 30, 37] and in adaptive security[33], where it is important to not only protect access to data, but also to operate on the data itself to prevent its inappropriate use, as when sensor images that contain some highly secure data (e.g., persons' faces, identified military objects) are 'fuzzed out' or 'blacked out' prior to distribution to others.

In summary, for any given data stream, the key question we ask is how to protect and secure certain data in that stream, distinguished by data type (e.g., 'passenger id' field of the 'passenger' event) and/or data content (e.g., data values and positions associated with face recognition). A second question is how to enforce such differential protection across multiple such streams in a distributed environment, where enforcement concerns the imposition limitations on certain stream manipulations by specific end users, as well as the ability to access specific stream data.

Security Overlays in Data Distribution Middleware. Our approach to attaining differential data protection augments data distribution middleware with additional security mechanisms, where security meta-information is automatically associated with the data being exchanged. Such meta-information is then used by middleware to guarantee that data is only accessible to and manipulable by authorized parties and that the manipulations by those parties are authorized as well. Essentially, we *overlay* onto existing data exchanges the security and protection currently needed. Security overlays are entirely dynamic, meaning that they can be changed and updated independently of the data streams they affect, where overlays may be altered while data exchanges are ongoing. The intent is to make security overlays as dynamic as the underlying systems being used and the applications being targeted.

Our current implementation of security overlays is in middleware running on standard operating system platforms. This implies that differential data protection is enforced only within the confines of the middleware infrastructure, and it requires that in addition to the data protection implied by security overlays, middleware must utilize authentication methods to ensure that data is not manipulated in unauthorized ways. The specific mechanism used is credentials, early examples of which are capabilities in systems like Hydra[23]. A credential is applied to some data stream, named by a *channel identifier*. This *credential* encapsulates a reference to a set of typed objects in the data stream and rights to these objects. The credential also serves to identify its bearer as an authenticated client. Based on the credential's meta-information (i.e., the type information), two actions may be taken with respect to the data stream. First, a *handler* may be applied to the stream, and the handler's operations can *extract* from the stream data of a certain type (e.g., of type 'passenger food preference') or *transform* the stream's data into a new form by applying computations to it (e.g., computing statistical information). Second, the newly created data can be *routed* to the client identified in the credential, the latter identified by a *client description*. This description currently contains an authenticated client identifier, but it can also use a more general way of identifying clients, such as trust levels, client roles, or group memberships (e.g., through community-based authentication[27, 5]).

A new data stream created by a handler is not actually produced until a *routing* action has been applied to it by some client authorized to receive this data. Such routing actions can be applied multiple times, by multiple authorized clients, thereby enabling the broad and flexible distribution of data required by the collaboration and remote sensing applications targeted by this work. The middleware implementation ensures that handlers and the data they produce are not executed unless the data is actually routed to some client. Routing actions can also be undone separately from the handler's creation of new data, thereby implementing a form of revocation[23]. Revocation can be performed by any client able to present a suitable credential.

Key Results. The key contributions of this work include the overlay mechanism for providing differential data protection to distributed, high-performance applications. Also, we show that such a mechanism can provide this level of protection functionality at little critical-path performance cost to the application. We show this by describing the effect of using the mechanism in a sample application described next.

2 Differential data protection in a distributed multimedia application

In this section we describe a representative application that can benefit greatly from differential data protection mechanisms.

2.1 The Active Video Streams application

We created Active Video Streams (AVS) to explore issues with deploying adaptive, high-performance data-streaming services in a distributed environment. By adaptive, we mean the ability of the application to react to changes in environmental conditions (network congestion, for example) or application-specific considerations (such as input from a human user). We also seek to provide such adaptive behavior at little or no cost in terms of run-time communication latency, bandwidth consumption, or more complicated costs associated with application re-engineering. Figure 1 shows the arrangement of the AVS application.

AVS consists of two components which communicate through the ECho [17] middleware system. The first component is a webcam driver that sends images along the communications link. An off-the-shelf webcam is used for this purpose, connected to a computer through a standard USB port. The webcam driver shares memory with the operating system USB driver. Images recorded from the camera are converted to PPM images and encoded with using a data-format system called PBIO [16]. Encoding data using PBIO provides a well-defined data structure for the middleware and application, as well as allowing AVS to ignore data heterogeneity issues (such as word-endianness).

Each data packet represents one frame from the video camera. The encoded data packets are then sent along an ECho event channel to the receiver component.

At the receiving end of the event channel, the other AVS component decodes the data from its wire-format representation using PBIO. AVS has two modules that can fill the role of the receiver component. One is a simple listener that writes frames to the local filesystem as they are received. The other is a Java-based viewer (the *player*) that provides the user with an interface on which each successive frame is shown. By using the viewer interface, the user effectively can see the real-time view picked up by the webcam.

Both receiving components allow the user to specify adaptive behavior that affects the data stream. For example, the user can choose to reduce the size of the image being streamed from the webcam driver to minimize network usage. These behaviors are implemented using ECho *derived* event channels. A derived channel is one that has an application-specified filter applied to it (Figure 2). These filters are provided as E-Code, a C-like language with modifications for safer computing (elimination of dynamic memory allocation and pointer manipulation, for example). The process of applying a modification to a stream is called *derivation* and a new event channel hosting the modified stream is created[1]. AVS handles the details of channel derivation without user intervention.

An example of such an application modification is the following. The default configuration for the webcam is to send RGB images. Suppose that a particular user wishes to instead see greyscale images from this webcam. This could be due to network conditions, where the 66% reduction in data being transferred might well improve performance. It is also possible that certain applications might better serve users if the image stream could be converted to greyscale on demand. The stream modification necessary to do the greyscale conversion is a piece of E-Code that performs a mathematical transformation on each pixel of the color image to produce a greyscale image.

2.2 Implications for security policy

The modifications described above have both clear and subtle implications for application security policy. Without a mechanism to provide data protection, the only policy that can be enforced is “anything goes”. In AVS terms, this means that anyone who can deduce an event channel identifier can access the image stream on that channel.

[1]For more information on E-Code and the creation of derived event channels, see [15].

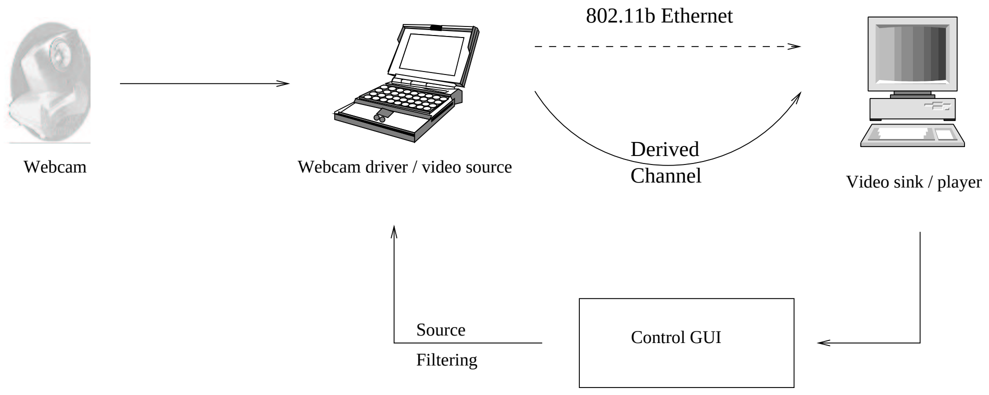


Figure 1. The Active Video Streams (AVS) applications consists of a camera driven by a separate host machine. This host machine serves as a video source and transmits images from the camera over a wireless communication link to a Java player application. The player application incorporates a control interface which can install filters on the event channel connecting the two hosts.

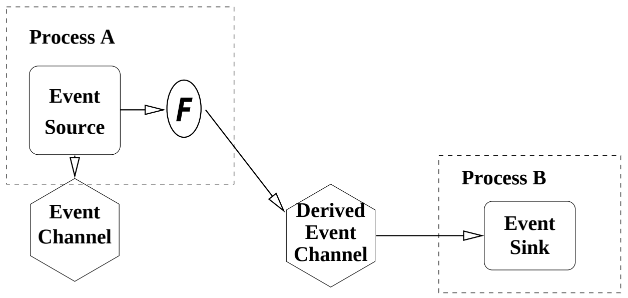


Figure 2. Derived event channels are created by applying a filter function to the output of an existing event channel.

"Access", in this context, can have several meanings. Suppose Mr. X is able to construct an event channel identifier. He might simply want to consume the event stream in the normal way, watching the video feed from the AVS webcam. For traditional access control and data protection scenarios, where access is restricted to the minimum required for any particular task, this is already an intolerable situation.

Mr. X is also able to tap the event stream and re-purpose it according to his own plans. Perhaps most alarming, the derived channel facilities described in the previous section can be used to introduce arbitrary modifications into the channel, including those designed to attack the stability of the application. In practical terms, a handler containing an infinite loop that is installed on a channel would effectively deny service to all clients of the channel.

The security policy statements suggested by Mr. X's potential actions are straightforward: "Only authorized users are allowed to consume data"; "No end user can install modifications on a channel"; "Only modifications C and D are allowed on this channel". Decoupling such policy statements from application code is also clearly preferable.

A flexible mechanism to implement security policy is also necessary. For example, publish/subscribe systems have attracted a great deal of attention due to the anonymity they afford their users - a publisher generally has no knowledge of the number or identity of subscribers. Publish/subscribe systems like AVS, however, clearly need the ability to differentiate between degrees and types of access. Meaningful security policy for AVS necessarily identifies at least classes of subscribers. Complicating matters is the dynamic nature of the application, where rapidly changing execution conditions (location of a mobile sensor, or time of day, for example) may dictate immediate security policy changes. A desirable mechanism should be able to implement general policy constraints ("User A lacks privilege to install the greyscale filter") without requiring the expression of those constraints in the application code. Also, such a mechanism is able to provide dynamic data protection as application policies change.

3 A model for differential data protection

In this section, we present a model for a protection mechanism that allows us to provide application-specific protection actions on structured data. These protection actions can be differentiated based on the structure of the data or other application considerations.

3.1 Foundations

We assume a system in which peered hosts exchange information using a publish/subscribe metaphor. In this system, *event channels* serve to transport information from host to host. The information flows between hosts can be stream- or packet- based, depending on application requirements. *Events* are discrete packets of information. A decentralized system of data types is applied to these packets. Hosts that do not recognize a particular type identifier can retrieve the definitions and any conversion information associated with that type either from a known location or from the sender. This allows us to assume a "global" type space without necessarily centralizing all type information.

We choose the event-based data exchange paradigm because of its importance to the class of applications targeted by this research. Event-based communications are widely used in the operational information systems used by companies like Delta Air Lines[26]. They are also the basis for large-scale information distribution systems like stock update notification[35]. Finally, they have been shown useful as a middleware basis for online collaborations in distributed engineering and science endeavors[28], and for the distributed sensor applications used as an example in this paper[2, 37].

3.2 Structure of credentials

A *credential* may be viewed as an abstract data type containing information needed and/or generated by the protection mechanism (Figure3 illustrates this). A credential contains an *object descriptor (OD)*, which uniquely identifies the object to which the information in the credential applies. This descriptor might be made concrete in an implementation as an object unique identifier, or, since we propose a system of types, as a combination of a instance UID with a type UID.

The other half of the classical capability format is a collection of rights. Rights govern operations that are possible for the object identified by the OD or on the credential itself.

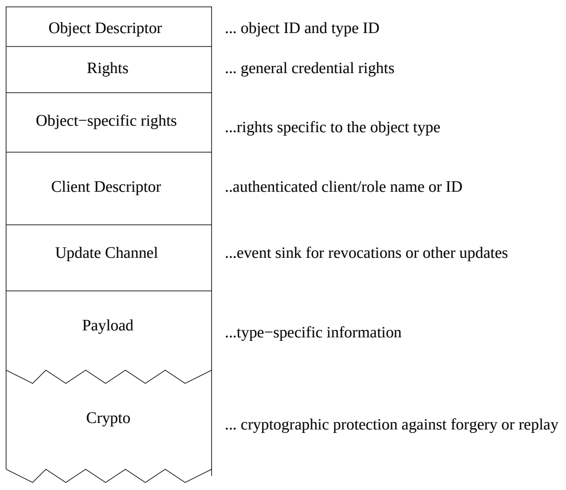

Object Descriptor
... object ID and type ID
Rights
... general credential rights
Object−specific rights
...rights specific to the object type
Client Descriptor
..authenticated client/role name or ID
Update Channel
...event sink for revocations or other updates
Payload
...type−specific information
Crypto
... cryptographic protection against forgery or replay

Figure 3. Credential structure detail.

Certain other special rights are indicated in the credential, like the ability to transfer ownership of a credential.

Ownership of credentials implies a mapping between credentials and owners. This mapping is represented by a *client descriptor (CD)* in the credential. The CD contains an authenticated (signed by a trusted entity) specific and unique client name. The entity named by the CD may take several forms. It may be a principal in a security policy matrix that represents a real-world person. It could also be the name of a security role, allowing role-based access control. Finally, the CD contains a trust-level designation, which may be used by authenticating entities to propagate trust or reputation characteristics.

Credentials contain a signed hash of their contents to guard against forgery. The signature can come from a trusted third party (one such party is described later) or from an arbitrary host in the system. In a case where an arbitrary host has signed the credential, the trust designation can be used to help make decisions about how reliable the contents may be.

The different object types to which credentials can refer are event channels themselves, source (submission) and sink (reception) handles to those channels, code segments (or references to them) used to extend or modify the behavior of event channels, and types and the conversions used to encode and decode data to and from transport format.

We are aware of the heavy reliance on cryptographic methods necessary to ensure the integrity of credentials. While highly-secure cryptography is expensive, our design minimizes the associated performance impact in two ways. We strive to keep cryptographic operations out of the "critical path" of information exchange as much as possible. For example, our mechanism makes it unnecessary to verify the integrity of a credential on every event submission or reception. Where cryptographic operations are necessary, we offer unique approaches designed to minimize their cost.

3.3 Routing

The publish/subscribe metaphor is anonymous in that event producers or sources do not necessarily know the identity, location, or number of event consumers. Overlays allow interested clients to *route* data streams to themselves by encapsulating the notion of subscription. Under the overlay mechanism, only authorized clients are permitted to subscribe to a channel. Credentials are used to indicate this situation - an authorized client is one that possesses a credential naming a particular channel in its object descriptor and the client in its client descriptor, and which has the route right indicated.

3.4 Handlers

Previous research [8] has demonstrated the effectiveness of associating code with event channels. This code can perform actions on the events passing through the channel or on the channel itself. Through this activity, event transmission can be efficiently customized for heterogeneous endpoints, dynamically varying network conditions, or application-specific purposes.

We call such code associated with an event channel a handler. In this position the handler can inspect each event passing through the channel. Since the events have specific types, the handler can perform a detailed, application-aware examination of the event. This allows the handler to perform actions based on the content of the event.

The event channel abstraction is designed to support anonymous subscription and data transfer. A motivating example for the development of AVS is the multiplexing of different levels of service across a single event submission from a source. The ability to do this reduces complexity at event sources. The anonymous nature of publish/subscribe systems is also preserved, as sources need not know which sinks are entitled which level of service. Flexibility in locating sources is also important, as low-power and -bandwidth constraints dictate that network transmission activity be kept to a minimum.

A typical AVS source is a webcam producing 640x480 color images. In this context, service level equates to stream quality - size, color depth, and frame rate are all axes along which AVS can differentiate service level. An obvious example where differentiation is desirable is an AVS instance where subscribers receive a basic level of service and must pay for higher service levels (or potentially special functionality). Other scenarios are equally valid, however - consider an instance where certain users are not allowed to view particular regions of the transmitted images.

A *transformation* is an overlay operation that allows applications like AVS to accomplish such tasks. Recall that the events exchanged by AVS are typed, with well-defined structure. Transforming handlers take events of one type and convert them to events of another type. A transforming handler, for example, can scale a 640x480 image down to half the original size for clients who desire or are only entitled to such images. Other image transformations implemented in AVS include greyscale conversion, image mirroring, and advanced transformations such as edge detection.

At the data structure level, a transforming handler receives a data structure as input, performs some computation on the contained data, and produces a structure of a different type as output. In the greyscale conversion case, a 640x480x3 (for RGB color) image is converted to a 640x480x1 image. The practical consequence of this is that the quantity of data output by the handler is one-third of the data the handler receives. Figure 4 illustrates this.

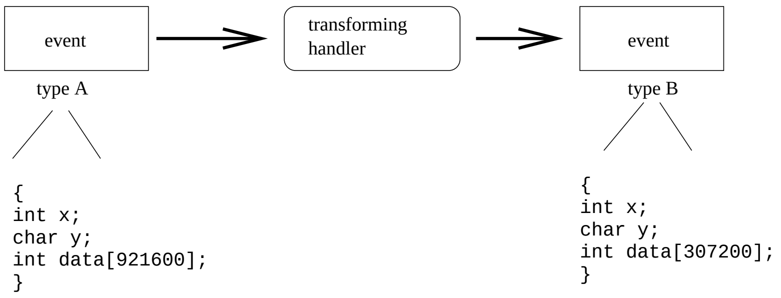
The overlay mechanism enforces the "assignment" of sink subscriptions to channels with appropriate transformation handlers (appropriate being an application-defined term here). Since credentials contain unforgeable references to channels (among other object types), a subscription requiring a particular credential implicitly restricts access to the image stream on that channel.

The transformation operation implies several rights-based operations. Since we have a quasi-global type system, we can make statements about whether a principal (really an entity identified by a client descriptor in a credential) can transform data to or from a type. Since channels are objects, we can also now talk about whether a handler can be installed on the channel that performs transformations on type of event being carried by the channel. A credential indicates these rights in its type-specific rights field.

3.5 Other infrastructure

The overlay mechanism relies on several other pieces of infrastructure; we briefly describe these here.

Security manager. Overlays require a trusted third party for several reasons. Most important is to issue credentials and vouch for their integrity. How such credentials are obtained is an open problem; we will refer to an outside agency called the Security Manager (SecMgr) which is the single point of contact for obtaining and revocation of credentials (as stated before, credentials are cryptographically secured against forgery). SecMgr's may choose to implement different security policies; credentials may be granted or denied based on challenge/response mechanisms like the UNIX login process, for example. It is the job of SecMgr to determine whether or not credentials are granted and for what period(s) of time they are valid;



```
{
int x;
char y;
int data[921600];
}
```

```
{
int x;
char y;
int data[307200];
}
```

Figure 4. Transforming handlers take one type of data as input and produce a different type as output. In this example, the handler converts the image data in the `data` array of the input type to greyscale from color, reducing its size by a factor of 3.

it is the job of the overlay mechanism, given properly obtained credentials, to ensure that no access except that specified by those credentials is allowed. This is the essence of the separation of policy and mechanism seen in systems such as Hydra that we apply to our own work.

Directory service. Client descriptors in credentials use globally unique names. This implies a global namespace and further that there needs to be some tool to coordinate access to the namespace. We rely on a directory service for this task. Applications can query the directory service to find information on objects or principals in the system.

Code repository. To install a particular handler, the location of an appropriate repository is retrieved from the directory service. This location is provided to applications in the form of a credential, with the object descriptor naming the repository and the client descriptor naming the application. The repository validates this credential and (with the cooperation and assistance of SecMgr) replaces the credential with one naming a specific piece of code.

4 Applying the model to AVS

We now describe how some of the concepts expressed in the overlay model have been realized in the AVS application. These concepts allow us to provide differential data protection to the AVS application. In turn, this differential protection allows the application to conform to various security policies without embedding those policies in application code. We also present empirical results that support our claims that the mechanism provides high-performance communication rates as well as the ability to perform differential data protection.

4.1 Supporting application-specific security policy

As stated earlier, a design goal of the overlay model is to support application-specific security policies without requiring application modification. We modified the AVS application to use a partial implementation of the overlay model. These modifications can be broken down into three parts: use of application policy descriptions, implementing a repository of handlers managed by secure infrastructure components, and code changes to the AVS application to use overlay credentials instead of direct references to the underlying middleware system.

4.1.1 Application policy descriptions

Our system uses XML [1] descriptions of security policies, with associated XML schemas to aid in policy definitions. We envision these descriptions being produced in several different ways: by graphical applications provided to end users, by automatic inspection of user databases and existing access control matrices, or by a local SecMgr instance in response to changing application or environmental conditions. XML provides a common, well-defined interchange format. We use HTTPS requests to retrieve them from standard web servers, providing both integrity of the policy descriptions and indirection to allow flexibility in policy deployment.

Note that since we are primarily interested in defining a mechanism capable of implementing a wide range of security policies, we are not concerned with issues such as reconciling conflicting policy statements, cached access to policy statements, or implementation of a general constraint engine. These issues are being addressed by others [11] and we rely on the interchange capabilities of our XML interface to provide interoperability.

For the AVS application, there are three types of policies:

those applied to the image regardless of what filter(s) might be installed, those governing access to image streams, and those governing access to filters for a particular stream. The first type of policy includes statements such as “the lower half of the image stream from camera A should be blurred” or “the lower half of all image streams viewed by user X should be blurred”. Access control policies for streams incorporate statements of the type “user X cannot access camera Y under condition Z”. The final type of policy statement includes statements such as “user X cannot install filter Y”.

Policy files for an application are provided to the SecMgr (via Uniform Resource Locator (URL)) when requests for credentials are made. The SecMgr, as described earlier, is responsible for generating and returning credentials as appropriate. Given appropriate credentials, the overlay mechanism will grant the indicated access.

Note that the mechanism may implement a policy by taking some action, not simply by granting or denying access. For example, assume that a particular camera views a sensitive area, and images are to be blurred in, say, the south-west quadrant. The mechanism prohibits any stream subscriptions without an appropriate credential. Appropriate credentials are only obtainable from the SecMgr, which requires the policy file containing the south-west quadrant restraint. Any AVS instance attempting to route the image stream (subscribe to it) must obtain a credential with route rights from the SecMgr. Any such credential issued by the SecMgr will contain an indication that a blur filter should be installed on the stream, but this will not be visible to the AVS instance requesting the route action. As far as the AVS instance is concerned, its route request succeeds and it gains access to a image stream. However, the blur filter that is implicitly installed (in response to the modified credential) blurs out the desired part of each transmitted image.

4.1.2 Filter repository

The filters used by AVS are described by XML documents residing on either the local filesystem of the receiving component or on an HTTP server accessible to it. A request for one of these filters is supplied by the player as a credential to the overlay mechanism. The credential is decoded to recover a URL indicating the location of the filter to be installed. The resulting XML document is retrieved, parsed to recover any associated metadata. The filter code may be contained directly in the XML document (usually if the filter is expressed in E-code), or the document may have a reference to the code repository (if the filter is a shared object to be loaded at run-time). Figure 5 shows how this process is accomplished.

4.1.3 Using credentials

The original implementation of AVS used direct references to objects of interest to the protection mechanism. These objects include names of communication channels and filters. The filters were directly embedded in the application code or could be provided directly by the user. These references and the code surrounding them were changed to use credential-based access exclusively. While this implies a number of additional network round-trips (for creation of credentials and retrieval of filters), the performance of the image stream itself is not affected. Since un-credentialed access to the image stream is not possible, the underlying performance of the AVS application is not significantly degraded.

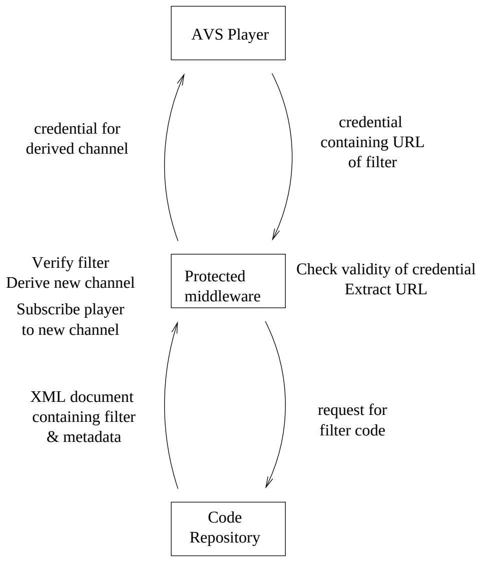

AVS Player
credential for derived channel
credential containing URL of filter
Verify filter
Derive new channel
Subscribe player to new channel
Protected middleware
Check validity of credential
Extract URL
XML document containing filter & metadata
request for filter code
Code Repository

Figure 5. Graphical depiction of the interaction between AVS, the protection middleware, and the code repository during the channel derivation process.

4.2 Experimental results

We profiled the modified AVS application against the original application in order to establish the amount of overhead the overlay mechanism imposes. For each middleware action, we recorded the percentage increase in time required to complete the action. Experiments were conducted using RedHat Linux 7.3 on a Dell Latitude C610 laptop computer as the video source (connected to the webcam), connected to the sink by a 802.11b wireless Ethernet link that feeds into the campus wired Ethernet. The video sink ran on

RedHat Linux 7.3 on a 600 Mhz PIII processor-based machine connected via 100Mbit Ethernet. The following table presents some representative results from these tests.

middleware operation	percentage overhead
channel create	4.52
channel subscribe	3.32
filter install	8.55
filter uninstall	3.33

The primary attractiveness of our mechanism is that its performance overheads are not in the critical path of data transfer. Once access to the channel has been established or a filter installed, data transfer proceeds at speeds limited only by the underlying middleware or network. Even in those situations where the mechanism does have a performance impact, the impact is minimal. We attribute the relatively long time taken to install a filter using the overlay mechanism to the network round trips necessary between the AVS application, the code repository, and the SecMgr. Furthermore, previous experience[7] has shown that using XML as a wire format leads to large network overheads. We believe that the interoperability advantages gained by using XML outweigh the performance impact, especially since the critical data transfer path is unimpeded.

5 Related work

Our conception of security overlays has most in common with the capability model proposed by Dennis and Van Horn and realized in Hydra[23]. In addition, we have adopted the principle of separation of policy and mechanism for the design of overlays. It is our intention that overlays be able to address a wide range of application security policies in a publish/subscribe environment. [11] presents a policy-definition architecture that is a useful example of the type of system overlays is designed to support.

The use of XML as a policy definition tool has also been a subject of previous work. [13] introduces a method of defining access restrictions on Web documents using XML. The Security Services Markup Language[25] proposes a method for expressing security models in XML. Additionally, although based on UML rather than on XML, QML[19] presented a method of describing system-level policies in an abstract fashion.

Several distributed computing and publish/subscribe approaches have at least partially addressed the issue of security policy and protection mechanisms. Grid computing approaches[27] are pursuing consensus on how to approach authentication and authorization issues over a widely distributed computing context. The Legion project[34, 18] is an object-based Grid computing infrastructure with a robust security architecture that features rights-based delegation of duties. We seek to develop solutions that are primarily applicable in a publish/subscribe context but that also may be useful for Grid researchers.

Finally, we note that the security and safety of a running application can be interpreted broadly as a quality-of-service issue. Our work bears similarity to BBN's Quality Objects[29] in its aspiration to apply a general policy (whether more traditional performance-based QoS or security) to a distributed system.

6 Conclusion

We have presented a method of providing differential data protection to applications through the use of a novel protected middleware mechanism. This mechanism, security overlays, provides applications with the ability to respond to high-level security policy information while preserving high-performance communication. We have also presented an application, Active Video Streams, that uses security overlays to implement a range of security policies. We have shown that the performance impact of our protected middleware layer is minimal, and occurs outside the critical path of data transmission.

Acknowledgments

This work was supported in part by an Intel Foundation Graduate Fellowship.

References

[1] The extensible markup language (XML). http://www.w3.org/TR/1998/REC-xml-19980210.

[2] The infosphere project. http://www.cc.gatech.edu/projects/infosphere.

[3] T. F. Abdelzaher and K. G. Shin. Qos provisioning with qcontracts in web and multimedia servers. In *IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999.

[4] A. Afework, M. Benyon, F. E. Bustamante, A. DeMarzo, R. Ferreira, R. Miller, M. Silberman, J. Saltz, and A. Sussman. Digital dynamic telepathology - the virtual microscope. In *Proceedings of the AMIA Annual Fall Symposium*, August 1998.

[5] D. Agarwal, M. Lorch, M. Thompson, and M. Perry. A new security model for collaborative environments. In *Proceedings of the Workshop on Advanced Collaborative Environments*, Seattle, WA, June 2003. LBNL-52894.

[6] M. Ahamad, G. Neiger, P. Kohli, J. Burns, and P. Hutto. Causal memory: Definitions, implementation, and programming. *Distributed Computing*, August 1995.

[7] F. Bustamante, G. Eisenhauer, K. Schwan, and P. Widener. Efficient wire formats for high performance computing. In *Proceedings of Supercomputing 2000*, November 2000.

[8] F. E. Bustamante. *The Active Streams Approach to Adaptive Distributed Applications and Services*. PhD thesis, Georgia Institute of Technology, November 2001.

[9] W.-W. Chen, , H. Towles, L. Nyland, G. Welch, and H. Fuchs. Toward a compelling sensation of telepresence: Demonstrating a portal to a distant (static) office. In T. Ertl, B. Hamann, and A. Varshney, editors, *Proceedings Visualization 2000*, pages 327–333, 2000.

[10] Y. Chen, K. Schwan, and D. Rosen. Java mirrors: Building blocks for remote interaction. In *Proceedings of the International Parallel Distributed Processing Symposium (IPDPS)*, April 2002.

[11] M. J. Covington, P. Fogla, Z. Zhan, and M. Ahamad. A context-aware security architecture for emerging applications. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, Las Vegas, Nevada, USA, December 2002.

[12] C. Cruz-Neira, D. Sandin, and T. Defanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *Proceedings of the SIGGRAPH 1993 Computer Graphics Conference*, 1993.

[13] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for xml documents. *ACM Transactions on Information and System Security*, 5(2):169–202, May 2002.

[14] A. Dey and G. Abowd. The context toolkit: Aiding the development of context-aware applications. In *Proceedings of the Workshop on Software Engineering for Wearable and Pervasive Computing*, Limerick, Ireland, June 2000.

[15] G. Eisenhauer, F. E. Bustamante, and K. Schwan. Event services in high performance systems. *Cluster Computing: The Journal of Networks, Software Tools, and Applications*, 4(3):243–252, July 2001.

[16] G. Eisenhauer, F. E. Bustamante, and K. Schwan. Native data representation: An efficient wire format for high performance computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(12), December 2002.

[17] G. Eisenhauer, K. Schwan, and F. Bustamante. Event services for high performance computing. In *Proceedings of High Performance Distributed Computing 2000 (HPDC 2000)*, 2000.

[18] A. Ferrari, F. Knabe, M. Humphrey, S. Chapin, and A. Grimshaw. A flexible security system for metacomputing environments. Technical Report CS-98-36, Department of Computer Science, University of Virginia, Charlottesville, Virginia 22093, USA, December 1998.

[19] S. Frolund and J. Koistinen. Quality of service specification in distributed operating systems design. In *Conference on Object-Oriented Technologies and Systems*. USENIX, April 1998.

[20] R. Herttwich. Keynote address at ARCS 2001.

[21] C. Isert and K. Schwan. ACDS: Adapting computational data streams for high performance. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, May 2000.

[22] C. Kasic and J. Walpole. QoS scalability for streamed media delivery. Technical Report CSE-99-011, Oregon Graduate Institute, September 1999.

[23] R. Levin, E. Cohen, F. Pollack, W. Corwin, and W. Wulf. Policy/mechanism separation in hydra. In *Proceedings of the 5th Symposium on Operating Systems Principles*, November 1975.

[24] J. López and D. O'Hallaron. Evaluation of a resource selection mechanism for complex network services. In *Proc. IEEE International Symposium on High-Performance Distributed Computing (HPDC10)*, pages 171–180, San Francisco, Aug. 2001.

[25] Netegrity. S2ML: The xml standard for describing and sharing security services on the internet. Technical report, 2001.

[26] V. Oleson, K. Schwan, G. Eisenhauer, B. Plale, C. Pu, and D. Amin. Operational information systems - an example from the airline industry. In *Proceedings of the First Workshop on Industrial Experiences with Systems Software (WEISS) 2000*, 2000.

[27] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks*. IEEE, 2002.

[28] B. Plale, G. Eisenhauer, K. Schwan, J. Heiner, V. Martin, and J. Vetter. From interactive applications to distributed laboratories. *IEEE Concurrency*, 6(2), 1998.

[29] C. Rodrigues, J. Loyall, and R. Schantz. Quality objects (QuO): Adaptive management and control middleware for end-to-end QoS. In *OMG's First Workshop on Real-Time and Embedded Distributed Object Computing*, Falls Church, Virgina, July 2000.

[30] D. I. Rosu, K. Schwan, S. Yalamanchili, and R. Jha. On adaptive resource allocation for complex real-time applications. In *18th IEEE Real-Time Systems Symposium, San Francisco, CA*, pages 320–329. IEEE, Dec. 1997.

[31] N. Sawant, C. Scharver, J. Leigh, A. Johnson, G. Reinhart, E. Creel, S. Batchu, S. Bailey, and R. Grossman. The tele-immersive data explorer: A distributed architecture for collaborative interactive visualization of large data-sets. In *Proceedings of the Fourth International Immersive Projection Technology Workshop*, Ames, Iowa, 2000.

[32] Schlumberger Limited. http://www.schlumberger.com.

[33] P. Schneck and K. Schwan. Dynamic allocation of security resources to client-server applications. In *IEEE Workshop on Dependable and Real-Time E-Commerce Systems*, Denver, Colorado, June 1998.

[34] G. Stoker, B. S. White, E. Stackpole, T. Highley, and M. Humphrey. Toward realizable restricted delegation in computational grids. In *Proceedings of European High Performance Computing and Networking (HPCN) 2001*, Amsterdam, The Netherlands, June 2001.

[35] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward. Gryphon: An information flow based approach to message brokering. In *International Symposium on Software Reliability Engineering '98 Fast Abstrac*, 1998.

[36] R. West, K. Schwan, I. Tacic, and M. Ahamad. Exploiting temporan and spatial constraints on distributed shared objects. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, Baltimore, Maryland, May 1997. IEEE.

[37] D. Zhou, K. Schwan, G. Eisenhauer, and Y. Chen. Jecho - interactive high performance computing with java event channels. In *Proceedings of the 2001 International Parallel and Distributed Processing Symposium*, April 2001.