A Flexible Security Model for Using Internet Content

Rangachari Anand, Nayeem Islam, Trent Jaeger and Josyula R. Rao
IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
{anand, nayeem, jaegert, jrrao}@watson.ibm.com

Abstract

Java applets, Netscape plug-ins and ActiveX controls have led to the popularization of a new paradigm: extensive downloading of executable code into applications to enhance the functionality of the desktop. One of the problems with this paradigm is the need to control the access rights of the downloaded content. In this paper, we describe a system for downloading content from the Internet and controlling its actions on a client machine. Our system generates a protection domain for the downloaded content dynamically using the content's requested domain and a policy database that describes the user's trust in the content's manufacturer and type. Our system ensures that this protection domain is enforced throughout the execution of the content. We have modified the Java Virtual Machine to implement our security model. Our implementation, called Flexxguard, is freely available at http://www.alphaworks.ibm.com.

1. Introduction

One of the promises of the World Wide Web is the ability of users to download content from arbitrary servers and then execute that content on their machines. From the viewpoint of a user, this paradigm is powerful because it decreases the amount of software that must be stored on the his/her machine and enables the creation of applications that can use custom components from vendors selected dynamically. From the content provider's viewpoint, it is easier to distribute content; all one has to do is put the content up on a web page. A web browser that supports the downloading of Java applets, Netscape plug-ins, or ActiveX controls is an example of this new computing paradigm.

A significant concern with this approach is that the downloaded software may be malicious and may damage the user's machine. The downloaded content can inflict damage on a user's machine when it has unrestricted access to the file system, network hosts and multimedia devices on the user's machine. One solution to this problem is to regard all downloaded software as hostile and to isolate it from all the resources on the user's machine. For instance, the current Java's solution is to either distrust all Java applets and disallow access to most resources on the user's machine and the network or completely trust the Java applet and allow it access to all local system resources and network hosts.

These two alternatives do not provide the level of flexibility required by emerging Internet applications, such as electronic commerce, groupware, workflow and games. Not only would each of these applications require access to different sets of resources on the user's machine, the user would like to personally configure the access an application has to resources on his/her machine. This is a classic engineering problem of mechanism versus policy: ideally, the user should be provided with the infrastructure necessary to support the *secure* execution of downloaded content, while being free to program a specific security policy. This model of security would be more flexible than the ones being currently supported by various interpreters and browsers and would enable a larger class of useful desktop applications.

2. Problem Definition

We assume an environment where an application downloads content from a potentially untrusted server over an untrusted network. A downloading principal is the principal that downloads, receives and executes the downloaded content. Downloading principals do not typically trust content manufacturers to not tamper with the system resources on their machines. For example, downloading principals would like to prevent content from: (1) reading private files; (2) writing executable files; (3) limit access to their system's CPU; and (4) prevent arbitrary remote communication from their system.

In addition to malicious content, *attackers* may also be present. An attacker is a powerful adversary that can read, modify, and delete any message sent between the content manufacturer and the downloading principal. Since an at-

1060-9857/97 $10.00 © 1997 IEEE

tacker can generate spurious messages, it must be possible to verify the source and integrity of the messages received by the downloading principal. In particular, the integrity of the messages containing the downloaded content should be verified. Since it is also possible for an attacker to eavesdrop on the communcations between the manufacturer and the downloading principal, privacy content messages may be required for some applications, such as electronic commerce. Finally, there should be some means of enforcing non-repudiability, that is, ensuring that a manufacturer cannot disavow its responsibility in producing the content.

The flexible control problem is to define and enforce a limited protection domain for downloaded content commensurate with its resource needs and the downloading principal's trust in the content's manufacturer. There are three facets to the control of downloaded control:

- **Authentication**: Verify the security requirements of the communication have been met such that the identity of the content's manufacturer and content can be determined or assume the content is generated by an untrusted source
- **Domain Derivation**: Derive a protection domain for this content
- **Enforcement**: Enforce the content's domain throughout its execution

Authentication is necessary to determine if the security requirements of the content message communication are fulfilled (e.g., integrity preserved). If the content message is authentic, then the sources of the downloaded content and the identity of the content can be determined. Alternatively, the content can be assumed to have been generated by an untrusted source.

Next, the protection domain for the content must be derived. For downloaded content, this domain can depend on: (1) the trust in the actual manufacturer and raters of the content; and (2) organizational policies. Finally, the content interpreter must be able to enforce the specified domain when the content is being executed.

3. Solution Overview

In this paper, we describe the design and implementation of a system that can flexibly control downloaded executable content. Flexibility is provided by: (1) the ability to execute both authenticated and unauthenticated content; (2) the ability to use statements about content made by trusted third parties, such as content rating services; (3) the ability to derive a protection domain dynamically for even new content; and (4) the ability to enforce different protection domains and resource limitations on different content executing simultaneously.

In our system, content can be delivered in encrypted form and with signed *stamps* provided by its manufacturer. A stamp is used to: (1) identify the manufacturer of the content and the principals involved in its distribution; (2) verify that no attacker has tampered with the content; (3) ensure that the manufacturer of the content cannot disavow his/her role in creating the content; and (4) enforce that no unauthorized eavesdropper can recover the content from the messages. We use public–key cryptography[20, 15], digital signatures and encryption to enforce this.

The stamp include information about the content's identity, purpose (e.g., using PICS ratings), and the resources it needs to execute on a user's machine. Content rating services can provide additional stamps that downloading principals can use to determine whether the content should be executed and the protection domain that the content should be granted.

A difficult challenge in these systems is to derive a "least privilege" protection domain for content. In our system, we permit a downloading principal and/or a systems administrator to configure a policy database with maximal protection domains for different content. Using the content identity information in the stamp, a maximal protection domain can be derived. This domain is compared to the requested domain provided in the content stamp to determine the content's actual protection domain. A downloading principal may optionally change the domain through a user interface (within limits).

The content's access to the resources on the client machine is regulated at runtime by a protection domain enforcer. It determines the content associated with the access and authorizes the access using that content's protection domain. Some denial-of-service attacks can also be prevented because the enforcer can monitor resource consumption.

4. Architecture

We propose a novel architecture for controlling the execution of downloaded content. In this architecture, downloading principals retrieve content and *content stamps* that attach descriptive information to content in a secure manner. Our architecture uses these content stamps to authenticate content and derive its protection domain. The content operations are then restricted to the derived protection domain. We define the following concepts in our architecture.

- The downloading principal: the client who is downloading the content and subsequently uses it.
- The manufacturer of the content: the principal that manufactured the content.

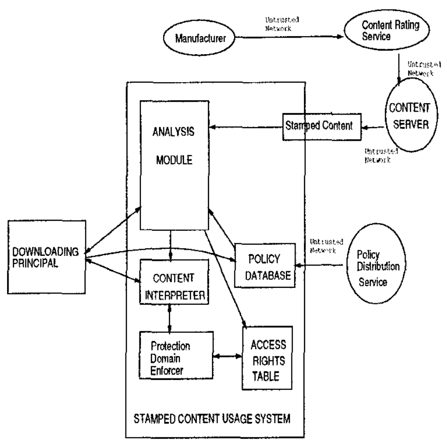

Figure 1. The Architecture of the System

- The content server: the server that delivers the content to the downloading principal;
- The certifying authority: A principal that manufactures public key certificates. These certificates bind the identities of principals to their public keys. The public key of the certifying authority is well–known.
- The policy distribution service: a principal that distributes policy databases to the downloading principal.
- The content rating service: a principal that rates content.
- The stamped content usage system (see Figure 4): (1) an analysis module that authenticates content and creates protection domains for content; (2) a content interpreter that interprets the content, (3) a policy database that holds the maximal domain associated with each principal; (4) a dynamic security enforcer that enforces access to system objects at runtime and (5) an access rights table that holds the capability lists associated with content.

The trust model of the architecture is defined from the perspective of the downloading principal. The downloading principal trusts certifying authorities, content rating services, the policy distribution service and the stamped content usage system. It does not trust the content server and the network. It has a varying degree of trust in the manufacturers of software.

We make the following assumptions about the environment in which our architecture is used. First, we assume that principals use a key distribution service that enables them to securely obtain another principal's public key. Next, we assume that all system I/O operations can be identified. This assumption depends on Java being a type-safe language. This has not been formally proven, and researchers have found errors in the verification and loading mechanisms that have enabled the type-safety to be circumvented[6]. However, we assume that the verifier enforces type-safety properly and do not address it further. Lastly, we assume that the operating system can be trusted. This is necessary to build a secure stamped content usage system upon it.

Our architecture solves the flexible control problem as follows (see Figure 4):

1. The manufacturer and content rating services upload content and/or content stamps to a content server,
2. The analysis module uses the content stamp to authenticate the downloaded content,
3. The analysis module uses the content stamp, the downloading principal's policy database (obtained from a policy distribution service), and perhaps some user intervention to derive the content's protection domain (set in the access rights table), and
4. The protection domain enforcer enforces the content's derived protection domain on any controlled operation performed by the content,

The content stamp specifies the authentication information and execution requirements of the content from the manufacturer's and/or rating service's viewpoint. The architecture is designed to utilize such information, as well as user input, to determine the content's protection domain. The domain enforcer must determine which content is being executed and enforce the appropriate protection domain upon it.

The subsequent subsections detail how the architecture performs the tasks listed above.

4.1. Content Stamp

Manufacturers and/or content rating services may create a *content stamp* to annotate content with authentication and execution information (analogous to PEM[2] or BETSI[16] certificates). Figure 2 shows various fields of the content stamp. In general, the content stamp is divided into two parts: (1) the *authenticator* which provides information for authenticating the stamp and the content and (2) the *tag* which provides descriptive information used to determine how to execute the content.

The authenticator consists of two fields. The first field, *security credentials list*, includes a hash of the content, a list of stamp signers, and their signatures of the stamp. These credentials are used to: (1) verify that the content

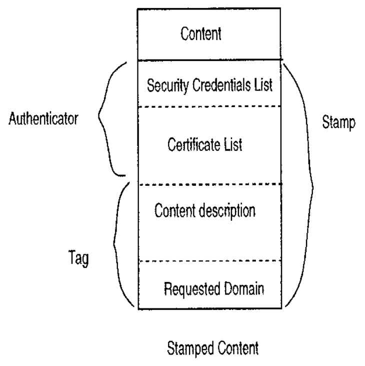

Figure 2. Structure of stamped content

was created and distributed by the principals whose credentials occur on the list; (2) verify that the content has not been altered after it was stamped; and (3) ensure that manufacturers cannot disavow the content that they have created. The separate content hash permits stamps to be downloaded separately from the content itself, if necessary.

The next field, *certificate list*, of the stamp contains a list of public key certificates (e.g., X.509) of the signer(s) of the stamp. For example, a stamp may be signed by a manufacturer and one or more content rating services. A public key certificate binds the identity of certificate's principal to its public key. A certifying authority signs such certificates.

The tag also consists of two fields. The first field, *content description*, contains information that describes the identity of the content, such as the name and version number of the software, the manufacturer's name, and the target execution platform. The second field, *requested domain*, specifies the protection domain that the content requests for executing the content. Content may need access to the following types of resources: the file system, memory, CPU, remote principals, and the downloading principal's display. Details of the protection domain specification syntax appears in [1].

When manufacturers create content, they may create a stamp for that content that binds their signature to the content's identity and requested domain. Downloading principals can use this stamp to verify that manufacturer signed the stamp, the content is unmodified, and that the manufacturer is the principal that vouched for the content and its requested domain (if the signature and hash verifications succeed).

Content rating services may sign manufacturer's stamps or create their own to annotate or revise the information in the manufacturer's stamp. For example, the content rating service may provide PICS rating information (see http://www.w3.org/pub/WWW/PICS/) about the content or revise its protection domain. The content rating services sign their stamps as well. Multiple content rating services may create stamps for the same content. For example, one may provide values for different rating attributes than the other. A content rating service is essentially a certifying authority (i.e., it is trusted by the downloading principal), so the content rating service's stamp with the same fields can be used to prove the same facts as a manufacturer's stamp. Therefore, only the content rating services stamp needs to be verified if both can be obtained.

4.2. Content Authentication

The downloading principal requests the content by sending a message to a content server. The content server receives the request and, if required, encrypts the stamped content (i.e., content and stamp) using either shared symmetric key, if one has been obtained (e.g., using Kerberos[17]), or the downloading principal's public key.

If a reply is not received from the content server within a specified timeout period the downloading process is terminated by the analysis module. Once the encrypted stamped content has been received by the downloading principal, the stamped content usage system's analysis module (see figure 4) authenticates the content to verify its integrity and source. It then decrypts the encrypted stamped content using the private key of the downloading principal. The public key of the manufacturer of the content or content rating service can be extracted from the public key certificate included in the stamp's certificate list or it can be obtained using the assumed key distribution service. The downloading principal uses the public key to verify that the stamp has not been modified. The analysis module computes a hash of the downloaded content and compares it to the hash in the stamp to verify that the content has not been modified. The identity information may also be used to verify that the content has the expected content name, version, platform, etc. If a stamp is provided, but cannot be verified, the associated content is not executed. Content that does not have a stamp is assumed to be from an untrusted principal.

4.3. Domain Derivation

Once the content has been authenticated, a protection domain must be computed for the content.[1] The protection domain determines the *access rights* the content has on the downloading principal's machine. We define protection domains using the following access control model. Access rights of a principal are a defined by a set of *domain rights* and *exceptions*.

- A domain right is a tuple <*system-object,allowed-ops,limit*> This states operations that are allowed on

[1] The system allows content that has no stamp to run in a highly restrictive domain.

an object and the limit on the number of such objects the operations are permitted upon,

- An exception is a tuple <*system-object,disallowed-ops*>. This specifies operations are not allowed on a system object.

A protection domain for principal is authorized to perform an operation on an object if: (1) at least one domain right permits the operation; (2) no exception exists that precludes the operation; and (3) the limit for the number of operations has not been exceeded.

The following inputs are used to derive content protection domains: (1) The requested domain (from the content stamp) (2) The maximal domain (from the policy database), and (3) User dialogue.

The following procedure is used to derive a content protection domain. An access right in the requested domain of the stamped content is added to the content's protection domain if it is within the maximal domain for the content as specified in the policy database. Other access rights may be approved by the downloading principal using a graphical interface. The protection domain in which the downloading principal may approve access rights for content may be restricted as well. The downloading principal may make these changes permanent by updating the policy database.

4.4 Policy Database

The policy database stores the mappings of content descriptions to default maximal protection domains. The analysis module can retrieve the default maximal protection domain for any content by supplying its description to the policy database. The policy database is organized to support partial matches of content descriptions to database entries, so default maximal protection domains can be retrieved for incompletely specified content and new content.

As Figure 3 shows, we use a three-level hierarchy for our policy database: general, manufacturer name, and content type. The general entry applies to all content. A manufacturer name entry applies to all content from the same manufacturer. A content type entry applies to all content of the same type from the same manufacturer or any manufacturer. Given D_{user} as the user's protection domain, the policy database represents the following relationships among protection domains: $D_{user} \supseteq D_{type} \supseteq D_{manufacturer} \supseteq D_{general}$. Additional levels, such as content name, can be added to extend the policy database. Also, different hierarchies can be specified for different users.

A default maximal protection domain is the union of the database entries whose key matches the content description. For example, the default maximal protection domain for *IBM Games* content is the union of the protection domains specified for the general, IBM, and *IBM Games* entries.

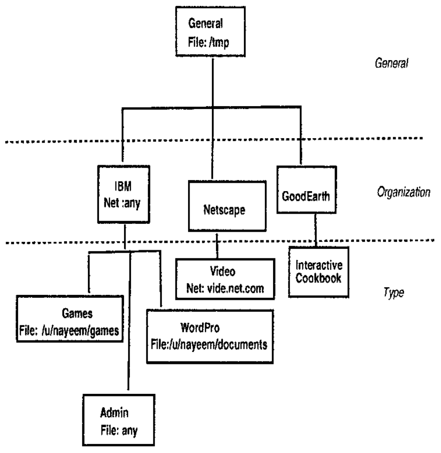

Figure 3. An example policy database hierarchy

However, the default maximal protection domain for IBM Financial Services content would be the union of the general and IBM entries only. Therefore, default maximal protection domains can be derived even for new content. Users can then extend the domain of the new content interactively.

A policy distribution service is defined to enable centralized management of security policy by system administrators. The service stores the policy database and responds to queries for default maximal protection domains and database modifications by users. We envision the policy distribution service being run on a secure machine. A secure channel is required to prevent unauthorized modification of the database. Access control to the policy database is enforced by associating ACLs with each database entry that specify who can perform read, write, delete, and specialize (i.e., create a child entry) operations.

When the stamped content is executed on the client machine, all operations on system objects are mediated by the protection domain enforcer. The protection domain enforcer authorizes such operations using the content's protection domain and monitors the resources consumed by the stamped content.

The protection domain enforcer stores the protection domain and resources consumed for all content in an *access rights table*. The access rights table has an entry for each stamped content that maps its execution identifier (e.g., thread id) to a tuple containing the content identifier attributes, signers, capability list, and runtime resource consumption (for limits). The identifier attributes and signers are used to recover the content identity. The capability list stores the access rights entries that comprise the content protection domain. The runtime resources consumed contains

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.