SUMMARY

As the reach and power of the Internet and networked systems widen, and thanks to the emergence of paradigm shifting technologies and delivery models like Web Services (WS) and Software as a Service (SaaS), ever larger numbers of users are sending huge amounts of private data to remote systems that they do not have any control over. On the other side of the same technology-coin, commercial digital content distributors are using the wide reach of the Internet to help disseminate digital content like music, videos and software to individual client machines, be it generic desktop machines or consumer appliances like multimedia players.

In general, these data and content providers have a strong interest in protecting their data from being misused. They would like their data to be used as specified by them, accessible only to explicitly allowed external parties and even after said access has been granted, allowing only specific actions to be performed on the data.

These access and usage specifications are usually expressed in the form of policies which can then be bundled with the data that they govern and sent over to the remote machines.

Several previous works have focused on how to express the restrictions that the policies define at the level of specification languages, while others have considered the problem from a more theoretical angle by formally defining models and classes of the policies that can be enforced based on various assumptions and capabilities of the system. Fewer works have, however, investigated the actual system level requirements involved in enforcing these policies on the remote machines. This is the angle from which we approach the problem in this dissertation.
PROBLEM STATEMENT

The broad problem statement that the work reported in this dissertation addresses can be expressed as follows:

*Given a data object that the user wishes to submit to an remote host and a policy that defines access and usage restrictions on the data object, design and implement an architecture that enables the enforcement of these policies at the remote host.*

While the development of a fully functional policy enforcement framework would involve several complementary areas of research including the policy expression language, policy modelling, formal analysis and the system architecture development, it is only the last of these research areas that form the subject of this dissertation. Where possible, existing works in the other areas are leveraged to fill in the gaps in the rest of the framework.

OUR APPROACH

The policy enforcement problem, as defined here, can be approached from various angles and using various levels of abstraction. Some of the earlier works and systems consider enforcing policies for specific applications or classes of applications. In these works the logic required to make enforcement decisions are built into the application code itself. Others, on the other hand, take a much lower level approach and consider the problem at the level of the operating system, exploring intricacies at the level of the operating system processes, describing which process can communicate with which other processes or access specific input or output channels.

Based on the level at which policy enforcement is performed, the classes of policies that can be interpreted (and hence enforced) also varies. Operating system level enforcement limits the enforcement classes to those that are readily describable at the process and system call level, while those at the level of specific applications confine themselves to those applications and their semantics alone.

In this dissertation, we consider the policy enforcement problem from a data-centric view point, assuming that the policy is attached to the data that are operated on by the applications. Our work approaches the problem at the middleware level, with the intention of exploiting the features of the higher and lower level solutions. This approach allows the architec-
ture to enforce data-specific, and not application specific, policies across multiple applications while at the same time not running the risk of losing application-semantic level information that would be valuable in enforcing a wider variety of policy classes. In particular we consider the enforce-
ment of policies for applications run in the Java Virtual Machine (JVM) environment. The rationale for this choice and details of the design of such an architecture are discussed in detail further on in this dissertation.

One of the key concepts that we leverage in our architecture is that of Information Flow Control (IFC), which deals with restrictions placed on how information can be transferred from one entity to another. While IFC as a research topic can be investigated from various angles, our work considers it from the perspective of the application’s programming lan-
guage semantics. Works in the area of IFC can be divided into two broad approaches—compile time and run time. In compile time systems, the in-
formation flow constraints are checked and verified at the time of compi-
lation. Run-time systems, on the other hand, perform these checks dy-
amically during the execution of the application. While each approach has its pros and cons, our architecture uses a hybrid approach, using the run-time mechanism enhanced with static control flow analysis, due to two main considerations: the ability of the enhanced run-time system to work without having access to the actual source code of the application and the larger classes of policies that can be enforced using this hybrid run-time approach.

THESIS CONTRIBUTION

In this dissertation we present the design, implementation and applica-
tion of a Java Virtual Machine based policy enforcement architecture. The contributions of this work are as follows:

- We examine in detail the previous work done in the problem space of policy enforcement and highlight the gaps our work aims to fill.

- We present the design and implementation of a JVM based informa-
tion flow control based middleware architecture aimed at enforcing policies associated with data objects.
– The middleware developed is used to implement an application independent Digital Rights Management (DRM) system using a widely studied usage control model as its basis.

– The JVM framework is also used to design a Web Service architecture capable of enforcing usage policies associated with the submitted data, as specified by the data provider.

One of the key functionalities introduced into this new JVM framework was that of information flow control, which allowed for very precise tracking of the data as they are used within the system as well as very flexible control over how the data can be used within the system. We also proposed a Java-like language for developing the core decision engine of Trishul.

The microbenchmark performance analysis performed on the just-in-time mode implementation of Trishul revealed areas of high overhead in the system. Subsequent analysis revealed design and implementation choices that could potentially decrease these overheads, which can be considered for potential future work.

In order to highlight the power and flexibility of the developed Trishul system, and to demonstrate that the system does indeed help in enforcing data-attached policies, we then used it as the building block of a policy enforcement architecture for two different application scenarios. In the first scenario, we designed and implemented a Digital Rights Management (DRM) system that was capable of enforcing several typical DRM policies attached to the multimedia content rendered by DRM applications running on top of the enforcement architecture. In the second application scenario, the Trishul framework was used to build a policy enforcement architecture for Web Services (WS), in which policies attached to the user data submitted to the WS were enforced by all the component services that make up the WS.

By verifying that the various component services run on top of the proposed enforcement architecture and in turn by designing and implementing the policy enforcement architecture using Trishul in a secure and extendable (modular) manner, we achieved the set objective of demonstrating the power and flexibility of the Trishul framework.
FUTURE WORK

There are a number of possible directions for future work, among others, open issues that have been identified.

The prototype implementation of Trishul, while ideal for showcasing the power and flexibility of the system to be used in various application scenarios, is far from an efficient implementation. As discussed in Section 3.5.4, several points of improvements have been identified, the implementation of which could constitute a direction for future work.

We have developed a Java-like Trishul-P language to allow policy engine writers to express the logic of the decision engines and hook it to the Trishul framework in an efficient manner. While the Java-like nature of the language lowers the barriers to adaptation of the system as well as makes it a powerful development tool, this same property makes it harder to theoretically analyse the expressibility of the language and the overall power of the enforcement system in terms of the classes of policies that can be enforced by it. The structured and logical expression of systems whose internal decision engine logic are expressed as automaton-based process steps make them ideal for structured analysis in comparison to the Turing-complete nature of Java-like languages.

Hence, a different direction of work would be to investigate either the implementation of Trishul-P with an automaton-based internal engine or the development of a front-end to the Trishul-P interface in the form of an automaton-based system. In the latter architecture, the policy engine writer would define the engine logic in the form of automaton transitions of allowed and disallowed states, which would then be transformed into Trishul-P code that, as before, can be compiled into loadable Java classes for use within Trishul. This allows for a more structured analysis of the power of Trishul architecture while at the same time making the development easier by reusing the existing solutions.

Trishul does not consider or address the impact of multi-threading in information flow or in application execution. Multi-threading introduces a set of new challenges and handling them could form a direction for future work.