An Architecture for Enforcing End-to-End Access Control Over Web Applications

Boniface Hicks, Sandra Rueda,
Dave King, Thomas Moyer, Joshua Schiffman, Yogesh Sreenivasan,
Patrick McDaniel, Trent Jaeger
Systems and Internet Infrastructure Security Laboratory
Department of Computer Science and Engineering
The Pennsylvania State University
boniface.hicks@email.stvincent.edu,
{ruedarod, dhking, tmmoyer, jschiffm, sreeniva, mcdaniel, tjaeger}@cse.psu.edu

ABSTRACT

The web is now being used as a general platform for hosting distributed applications like wikis, bulletin board messaging systems and collaborative editing environments. Data from multiple applications originating at multiple sources all intermix in a single web browser, making sensitive data stored in the browser subject to a broad milieu of attacks (cross-site scripting, cross-site request forgery and others). The fundamental problem is that existing web infrastructure provides no means for enforcing end-to-end security on data. To solve this we design an architecture using mandatory access control (MAC) enforcement. We overcome the limitations of traditional MAC systems, implemented solely at the operating system layer, by unifying MAC enforcement across virtual machine, operating system, networking and application layers. We implement our architecture using Xen virtual machine management, SELinux at the operating system layer, labeled IPsec for networking and our own label-enforcing web browser, called FlowwolF. We tested our implementation and find that it performs well, supporting data intermixing while still providing end-to-end security guarantees.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and protection; D.4.6 [**Operating Systems**]: Security and Protection—*access controls*

General Terms

Security, Virtual Machines

Keywords

Access Control, Xen Security Modules, Policy Compliance

1. INTRODUCTION

The web has evolved into a general purpose distributed computing platform. Wikis, bulletin board systems, collaborative editing environments, search engines, calendars, and many other web applications handle data that is obtained from and shared with large bodies of loosely associated hosts. In a wiki, for example, data originating from one user may be stored on a web server and then mixed with another user's data within that user's web browser. This rich intermixing of data from many sources allows users to create new, innovative models of data sharing, e.g., mashups.

A cost of this intermixing of data is client and server vulnerability to a broad milieu of attacks. For instance, XSS attacks that execute malicious scripts on an unsuspecting user's browser. Scripts that corrupt or leak sensitive data (passwords, credit cards, etc.) Such a vulnerability is a consequence of a fundamental limitation of the existing web infrastructure: existing environments provide no means for enforcing end-to-end security. As in the XSS attack, the lack of coordination between elements of the system allows the adversary to abuse the system—the fact that the malicious script originated from a potentially dangerous input is not known by the user, i.e., it is delivered in the same "security context" as the legitimate content.

Efforts to secure web applications have historically focused on client-side solutions. These can be divided into two categories: inter-browser separation and intra-browser separation. On the inter-browser side are systems, such as Tahoma [**?**] or NetTop [**?**], that use separate virtual machines to separate data with different security requirements. On the intra-browser side are various systems such as OP's process-based separation of plugins [**?**] and Chrome's process-based separation of tabs, as well as more fine-grained approaches like the same-origin policy. While these solutions provide additional security and protect against some attacks on the browser side, none are able to coordinate the management of data that is *necessarily intermixed* within the same browsing context. What is needed is a way to integrate the security enforcement at each layer of the system—thereby ensuring that mixing is consistent with the security policies of all elements of the system.

Our solution is to use mandatory access controls (MAC) to enforce an end-to-end security policy on web application data. It is insufficient to follow the traditional approach to MAC, however, which focuses on implementing MAC only at the operating system layer. This approach is too restrictive, preventing the rich intermixture of data from various sources that must take place in the web browser. Rather, we find that by unifying MAC enforcement across the virtual machine, operating system, networking, and application layers we can achieve more effective and more flexible enforcement than the operating system alone.

In this paper, we provide an architecture that implements a multi-layer, mandatory access control system for enforcing end-to-end security goals on web applications. Data originating on the server side is labeled by the web application administrator and data originating on the client side is labeled by the user. The system is configured such that it propagates labels with data across each layer of the system—from the server to the Virtual Machine (VM) to the Virtual Machine Manager (VMM) to the network layer to the client and back again. Each layer is configured such that it consistently enforces policy on labeled data with respect to the other layers.

We implement our architecture using a Xen VMM, SELinux VMs, labeled IPsec and our own instrumented label-enforcing browser called FlowwolF, along with some other custom-built components for supporting policy distribution and distributed label mapping. The major challenges we had to overcome in the implementation were augmenting a web browser to enforce mandatory policies, defining comprehensive policies for web applications covering each layer, automatically distributing policies across layers to appropriate enforcement points and caching policies appropriately to achieve performance levels that would not diminish the user experience expected of web applications.

This paper makes the following contributions:

- We develop the first architecture for a multi-layer, mandatory access control web application system;
- We provide a working implementation of the system using commodity as well as custom components;
- We implement a label-enforcing web browser that gives and receives labels from the system;
- We instrument an open source bulletin-board messaging system such that it enforces end-to-end security goals when used in our system.

The rest of the paper is organized as follows. In Section 2 we present the problems we must solve when developing an end-to-end secure web browsing system. In Section 3 we give some background on mandatory access control systems with the requirements a system must fulfill to get the security guarantees these systems provide. We follow that in Section **??** by proposing an architecture using a mandatory access control system that solves these problems. In Section **??** we describe our implementation of the architecture. In Section **??**, we evaluate the architecture. To do so, we instrumented a commodity bulletin board application. Finally we conclude in Section **??**.

2. WEB APPLICATION SECURITY

To deliver rich, dynamic content, today's Web 2.0 applications combine data from sources with varying security requirements and render them in a single browser tab. Many popular applications such as Mashups and social networking sites allow other applications to read and write data or act as the user. If the applications trusted to perform these actions are malicious, they could leak or destroy sensitive data.

To illustrate, consider our instrumented Bulletin Board application running in a multi-level secrecy (MLS) environment [1]. In this case, the posted messages have security requirements that originate at the client side (in the browser). A user in the MLS environment is gicen a set of security labels when she logs into her OS (her clearance). These labels must be checked by the browser before allowing her to open a secret browser tab to read and write secret data. Before she can post, the browser must ensure she is posting from a secret tab and the content of her message was entered only from secret input fields (not replying to a top-secret message, for example). Furthermore, the system must ensure that the remote server she is posting to is secret, such that she can open a secret socket that is connected to a secret web server application running on a remote OS that protects its applications' secrecy. This opens up various attack vectors. We divide these attack vectors into three categories—network-layer attacks, OS-layer attacks and browser-layer attacks.

[1]Our approach is not specific to MLS, but it will require a mandatory access control policy, see Section 3.

Network-layer attacks focus on the interception of secret data and hijacking connections to impersonate trusted authorities. One example, a man-in-the-middle attack (MITM), allows an attacker to act like a proxy for the browser, intercepting web requests and providing the browser with the content returned from the server. The attacker can then modify the message content or steal secret messages. More simply, if a network channel is not encrypted, it is subject to modification by injection or leakage by eavesdropping.

At the OS layer, secret posts stored on the Bulletin Board server or stored persistently in the web browser's system are vulnerable to attacks by any malware running on the system. Malware could modify the web browser and server application by modifying runtime libraries, leaking secure message data or causing other mischief that undermines the message security requirements enforced by web browser and server.

At the browser layer, there are myriad attacks that involve compromising the browser or confusing it (or the user) to behave incorrectly. If public and secret messages are loaded into the same browser tab, malicious scripts in public messages might leak secret messages to an attacker's site (i.e., *cross-site scripting*, or XSS). Other attacks like *cross-site request forgery* (CSRF), can trigger a request on behalf of the user without their knowledge. This script then uses the user's authenticated credentials to access secrets for the attacker. OWASP listed these as the most common browser vulnerabilities [**?**]. Another prevalent danger is *phishing*: innocent-looking public messages can solicit viewers to click on links that would send secret data as parameters to malicious sites or request that a user fill out a form that will post secret information to malicious sites. Next we describe the various models developed that employ isolation techniques to prevent data leakage and compromise.

VM-layer or inter-browser separation Approaches such as NetTop [**?**] and Tahoma [**?**] separate data sources of different security levels into individual virtual machines (VMs). It has the advantage of providing strong separation and preventing many of the attacks we have described. Browser level attacks like XSS and CSRF attacks are prevented because access to untrusted domains are prohibited. However, this approach reduces efficiency and violates our usability requirements. In our bulletin board example, a user with secret clearance would have to view different messages in different VMs rather than viewing multi-level messages in the same browser or even the same tab. Furthermore, a flexible use model would allow a single browser to access multiple trusted web applications without total separation such that they could modify each others' data in controlled, policy- driven ways. Finally, these approaches alone fail to protect against attacks at the network layer (like MITM) or the OS-layer attacks, like those from malware.

Intra-browser separation In order to maintain the usability advantage of displaying data from various origins in a single page, recent research attempts to separate data of different secu-

Layer	*Protection State*	*Labeling State*	*Transition State*	*Enforcement*
Application	DLM restrictions	Data	Low to high secrecy	Control data leakage
VM	SELinux policy	Processes	Within VM label range	Prohibit loading illegal security level browsers
VMM	XSM policy	VMs	Spawning VMs	VM ranges limited to policy
Network	IPsec policy	Sockets	Single level	Prevents connection with insecure remote machine

Table 1: Elements of the mandatory protection system needed at multiple layers for end-to-end secure web systems.

rity levels using isolation policies within the browser. For example, the same origin policy prevents Javascript from one origin from modifying Javascript data originating elsewhere. Other innovative approaches refine this policy to prevent some attacks by requiring some mutual authentication [?], by marking up DOM content with accents [?], instrumenting the Javascript interpreter mediation points [?, ?], or performing client or server filtering to remove malicious code [?]. These approaches have the advantage of minimal modifications to the system, modifications to the client-side alone, and incremental deployment. A more robust approach to intra- browser separation introduces process separation to protect the browser from malicious plugins [?]. None of these approaches, however, provides protection against phishing, network-layer or VM-layer attacks and only provides partial protection against browser attacks.

Most of these intra-browser approaches suffer from not knowing precisely or confidently the security properties of web data. They impose a heuristic policy, presuming for example, that data from the same origin should have the same security properties. This assumption would be false like in our bulletin board example, which serves up both public and secret data on the same site.

None of the above approaches alone is adequate for systems that seek to enforce strong, end-to-end security goals while still combining data of mixed security in a single browser window. What is needed is a web application system that combines the previous two approaches, using inter-browser security techniques for protection against OS-layer and network-layer attacks as well as intra-browser protections for improved usability and protection against XSS, CSRF, and drive-by download attacks. To maintain security policies on data between server and browser, protection is needed at each layer: the application (intra-browser) layer, VM layer, VMM layer and network layer. Intra-browser separation can be improved by having authenticated security policies on secret, trustworthy data, and network-layer and OS-layer attacks can be prevented by enforcing mandatory policies on web application data and programs.

Designing such a web system introduces a host of challenges. One challenge is properly configuring the system to statically and dynamically label web content while propagating those labels faithfully. Another is carefully dividing up the label enforcement between layers while ensure security requirements are maintained end to end. Furthermore, this must be done efficiently.

3. END-TO-END ENFORCEMENT

We claim that a system provides secure end-to-end enforcement for an application if a mandatory access control (MAC) policy is enforced consistently across all software layers. Anderson defined the *reference monitor concept* [?], which states the guarantees that must be satisfied to enforce a MAC policy correctly. We propose the construction of a multi-layer reference monitor for end-to-end enforcement. Table 1 shows the system layers (in rows), the MAC policy concepts (in columns), and the requirement assignment of MAC policy to layers (in each cell). Our task is to define a multi-layer reference monitor that enforces a coherent system-wide MAC policy and demonstrate what is necessary to build it correctly (Section 3.1). We also define a mandatory protection system, which motivates why a MAC policy is necessary for our multi-layer reference monitor and identifies the MAC policy concepts that must be enforced (Section ??).

3.1 Multi-Layer Reference Monitor

Table 1 shows four layers needed to enforce a MAC policy: (1) the *application layer* controls access to application objects (e.g., browser tabs and URLs); (2) the *VM layer* consists of the operating system that controls process (including the application) access to VM system objects (e.g., files and sockets); (3) the *VMM layer* (e.g., for Xen, its hypervisor and privileged host VM) controls inter-VM interactions (e.g., shared memory and communications); and (4) the *network layer* that authorizes communication and dictates how secure communication is performed (e.g., chooses cryptographic protocols). We state that there may be multiple *components* at each layer that are required to enforce a MAC policy, such as multiple application processes or multiple VMs. Each of the components that we depend upon to enforce a MAC policy must be part of the multi-layered reference monitor for that application.

A true reference monitor must satisfy [?]: (1) *complete mediation*—all security-sensitive operations must be mediated by the reference monitor; (2) *tamperproofing*—the reference monitor must be protected from illicit modification; and (3) *simple enough to verify*—the reference monitor mechanisms and policies must be verified to enforce site security goals correctly. A multi-layer reference monitor must ensure that the composition of layers satisfies these requirements. Below, we examine these requirements and the tasks that must be performed to satisfy them in a layered environment.

We leverage existing mediation in OSs (e.g., SELinux [?]) and VMMs (e.g., Xen sHype [?]), but we also require mediation for other layers and an approach for inter-layer communication of security information. For the application layer, we extend the browser application with mediation mechanisms that leverage OS labels. We use labeled IPsec [?] to authorize access at the network layer using system labels that specifies secure communication requirements. Additionally, we use the same set of system labels for all layers for consistence across layers.

Tamperproofing each layer is generally done by the layer below. For example, an SELinux policy must protect the browser process from tampering by other processes if it is to enforce its policy correctly. In prior work, we evaluated SELinux policies to ensure that reference monitoring processes cannot be tampered with by untrusted processes [?]. As the browser application is just another instance of a reference monitoring application, the same technique can be used, so we do not discuss this further. Ensuring that a VMM policy protects a VM from tamperproofing can be performed similarly.