A Theory of Runtime Enforcement, with Results

Jay Ligatti and Srikar Reddy

University of South Florida
Department of Computer Science and Engineering
{ligatti,sreddy4}@cse.usf.edu

Abstract. This paper presents a theory of runtime enforcement based on mechanism models called MRAs (Mandatory Results Automata). MRAs can monitor and transform security-relevant actions and their results. Because previous work could not model monitors transforming results, MRAs capture realistic behaviors outside the scope of previous models. MRAs also have a simple but realistic operational semantics that makes it straightforward to define concrete MRAs. Moreover, the definitions of policies and enforcement with MRAs are significantly simpler and more expressive than those of previous models. Putting all these features together, we argue that MRAs make good general models of runtime mechanisms, upon which a theory of runtime enforcement can be based. We develop some enforceability theory by characterizing the policies MRAs can and cannot enforce.

Keywords: Security models, enforceability theory.

1 Introduction

Runtime enforcement mechanisms work by monitoring untrusted applications, to ensure that those applications obey desired policies. Runtime mechanisms, which are often called runtime/security/program monitors, are quite popular and can be seen in operating systems, web browsers, spam filters, intrusion-detection systems, firewalls, access-control systems, stack inspection, etc. Despite their popularity and some initial efforts at modeling monitors formally, we lack satisfactory models of monitors in general, which prevents us from developing an accurate and effective theory of runtime enforcement.

1.1 Related Work

It has been difficult to model runtime mechanisms generally. Most models (e.g., \[16\][18][9][11][5]) are based on truncation automata \[16\][12], which can only respond to policy violations by immediately halting the application being monitored (i.e., the target application). This constraint simplifies analyses but sacrifices generality. For example, real runtime mechanisms often enforce policies that require the mechanisms to perform “remedial” actions, like popping up a window to confirm dangerous events with the user before they occur (to confirm a web-browser connection with a third-party site, to warn the user before
downloading executable email attachments, etc). Although real mechanisms can perform these remedial actions, models based on truncation automata cannot—at the point where the target attempts to perform a dangerous action, truncation automata must immediately halt the target.

To address the limitations of truncation automata, in earlier work we proposed edit automata, models of monitors that can respond to dangerous actions by quietly suppressing them or by inserting other actions \[12\]. By inserting and suppressing actions, edit automata capture the practical ability of runtime mechanisms to transform invalid executions into valid executions, rather than the ability of truncation automata to only recognize and halt invalid executions. Edit automata have served as the basis for additional studies of runtime enforcement (e.g., \[19\][17][14]).

Unfortunately, while truncation automata are too limited to serve as general models of runtime mechanisms, edit automata are too powerful. The edit-automata model assumes monitors can predetermine the results of all actions without executing them, which enables edit automata to safely suppress any action. However, this assumption that monitors can predetermine the result of any action is impractical because the results of many actions are uncomputable, nondeterministic, and/or cannot tractably be predicted by a monitor (e.g., actions that return data in a network buffer, the cloud cover as read by a weather sensor, or spontaneous user input). Put another way, the edit-automata model assumes monitors can buffer—without executing—an unbounded number of target-application actions, but such buffering is impractical because applications typically require results for actions before producing new actions. For example, the echo program \[x=input(); output(x)\] cannot produce its second action until receiving a result, which is unpredictable, for the first. Because the echo program invokes an action that edit automata cannot suppress (due to its result being unpredictable), this simple program, and any others whose actions may not return predictable results, are outside the edit-automata model.

1.2 Contributions

This paper presents a theory of runtime enforcement based on mechanism models called MRAs (Mandatory Results Automata). Their name alludes to the requirement that, unlike edit automata, MRAs are obligated to return a result to the target application before seeing the next action it wishes to execute. In the MRA model, results of actions may or may not be predeterminable.

Conceptually, we wish to secure a system organized as in Figure \[1\], with an application producing actions, and for every action produced, the underlying executing system (e.g., an operating system, virtual machine, or CPU) returning a result to the target application. Results may be exceptions or void or unit values, so all actions can be considered to produce results. For simplicity, this paper assumes all actions are synchronous; after the application produces an action \[a\], it cannot produce another action until receiving a result for \[a\]. In contrast, the edit-automata model can be viewed as one in which all actions