# Information Flow Containment: A Practical Basis for Malware Defense

R. Sekar



# Information Flow Containment: A Practical Basis for Malware Defense*

R. Sekar

Stony Brook University

Security threats have escalated rapidly in the past decade. "Zero-day attacks," delivered via web pages, pictures or documents, have become significant threats. Malware is rampant, being installed using phishing, software vulnerability exploits, and software downloads. With the emergence of a lucrative black market in cyber crime, even ordinary users are becoming targets of sophisticated malware attacks.

Existing malware defenses rely mainly on *reactive* approaches such as signature-based scanning, behavior monitoring, and file integrity monitoring. Malware writers are increasingly deploying code obfuscation to fool signature-based detection. They can also modify malware behavior to fool behavior-based techniques. Moreover, to further complicate the development of signatures or profiles, malware is increasingly incorporating anti-analysis and anti-virtualization measures. Finally, sophisticated malware uses rootkit-like techniques to hide its presence from virus scanners and file integrity checkers.

The most commonly deployed *proactive* defense against untrusted (and hence potentially malicious) software is behavior confinement, i.e., restricting access permissions of software using restrictive, fine-grained access control policies. Policies may be enforced on code downloaded from untrusted sources, as well as processes such as web browsers that are at high risk of being compromised. Untrusted processes may be restricted by these policies in terms of their access to system resources (e.g., files) and inter-process or inter-host communication. Unfortunately, an adversary that knows the policy can easily modify their malware so that it can achieve its goals without violating the policy. For instance, if a policy prevents an untrusted process from writing files in system directories, it may simply deposit a shortcut on the desktop with the name of a commonly used application. When the user subsequently double-clicks on this shortcut, malware can do its work without being confined by a policy. Alternatively, malware may deposit files that contain exploits for popular applications such as those used for creation or viewing of documents and pictures, with the actual damage inflicted when a curious user opens them. Indeed, there are numerous ways to mount such multi-step attacks, and it is very difficult, given the complexity of today's applications and operating systems, to eliminate every one of them. Of course, it is possible to impose very restrictive policies, such as preventing any file writes, but this will come at the expense of usability and will likely be rejected by users.

* This work was supported in part by ONR grants N000140110967 and N000140710928, NSF grants CNS-0208877 and CNS-0831298, and AFOSR grant FA9550-09-1-0539.

A key feature of many malware infections, including the multi-step attacks described above, is the subversion of legitimate (also called *benign*) processes that aren't confined by strict policies. Thus, *rather than focusing on untrusted process confinement,* our research focus has been on *isolating benign processes* from untrusted data and code. In addition to restricting the execution of untrusted code by benign processes, our approach also restricts benign processes from consuming any data that resulted (in part or whole) from an untrusted process. As a result, there can be no causal relationship between the actions of a benign process and those of untrusted malware.

One approach we have developed is based on the concept of *one-way isolation,* where information can flow freely from benign applications (or data) to untrusted applications, but the reverse flow is blocked. In particular, all data created or modified as the result of executing an untrusted application are contained within our *safe-execution environment (SEE),* and is inaccessible to benign applications. SEEs are not only suitable for trying out untrusted software, but have several other interesting applications, including testing of software patches and upgrades, penetration testing, and testing out new software configurations. Our SEE enables these tasks to be performed safely, and without disrupting the operation of benign servers and desktop applications that are running outside the SEE. Moreover, if the result of an SEE execution is determined to be safe by an user, he or she may *commit* the results so that they become visible to the rest of the system. We have developed simple and effective criteria to ensure system consistency after a commit.

Although our SEE is effective in restricting information flows without affecting the usability of untrusted applications, there is one problem it cannot solve by itself: users need to decide whether the results of untrusted execution are "safe" to be committed to the host system. We have explored ways to automate this step. In its most basic form, this automation is achieved by encoding the safety criteria in the form of a program, and by permitting this (trusted) program to examine the state inside the SEE. If the SEE state is determined to be safe, then its contents are committed, as mentioned before. We point out that a policy enforcement mechanism that combines isolated execution with post-execution state examination is more powerful and flexible than a traditional behavior confinement mechanism. In particular, behavior confinement policies need to be written so that every permitted operation leaves the system in a safe state. In contrast, our hybrid approach allows the system to go through intermediate states that are unsafe. For instance, we can permit an execution that deletes a critical file and recreates it, provided the recreated content is equal to the original content (or contains some permitted modifications). In contrast, a traditional behavior confinement system would require aborting the execution at the point the application attempts deletion of the critical file.

We then considered the special but important case of verifying the safety of software installations. Since software installations normally require high privileges, they are a favorite target for malware writers. If malware can trick a user into permitting it to be installed, then, by utilizing the administrative privileges

that are available during the installation phase, malware can embed itself deeply into the system. We have developed an approach that can automatically identify the correctness criteria for an untrusted software installation, and verify it after performing the installation within an SEE. Our technique has been implemented for contemporary software installers, specifically, RedHat and Debian package managers.

Most recently, we have been investigating an approach that performs comprehensive information-flow tracking across benign and untrusted applications. The advantage of such an approach is that it can altogether avoid the question of what is "safe." Instead, data that is produced (or influenced) by untrusted applications are marked, and any process (benign or untrusted) that consumes such data is confined by a policy. Moreover, outputs of such processes are also marked as untrusted. Although the concept of information-flow based integrity is very old, its practical application to contemporary operating systems has not had much success. Guided by our experience with SEEs, we have developed an effective and efficient implementation of this approach for contemporary operating systems, specifically, recent versions of Ubuntu Linux. This talk will conclude with a description of our approach, and our experience in using it.