Standard access control mechanisms are often insufficient to enforce compliance of programs with security policies. For this reason, information flow analysis has become a topic of increasing interest. In such type of analysis, the main property to be checked is called non-interference, which basically states that the publicly observable behavior of a program is entirely independent of its secret, secure input values.

However, simple non-interference is too restrictive for specifying and enforcing information flow policies in most programs. Exceptions to non-interference are provided using declassification policies. Several approaches for enforcing declassification have been proposed in the literature. In most of these approaches, the declassification policies are embedded in the program itself or heavily tied to the variables in the program being analyzed, thereby providing at best little separation between the code and the policy. Consequently, the previous approaches essentially require that the code be trusted, since to trust that the correct policy is being enforced, we need to trust the source code.

In this thesis, we propose a novel framework for information flow analysis, with support to declassification policies, related to the source code being analyzed via its I/O channels. The framework supports many of the declassification policies identified in the literature. Based on flow-based static analysis, it represents a first step towards a new approach that can be applied to untrusted and legacy source code to automatically verify that the analyzed program complies with the specified declassification policies. We present a framework in which expressions over input channel values that could be output by the program are compared to a set of declassification requirements. We build an implementation of such framework, which works by constructing a conservative approximation of the such expressions, and by determining whether all of them satisfy the declassification requirements stated in the policy. We introduce a representation of such expressions that resembles tree automata. We prove that if a program is considered safe according to our analysis then it satisfies a property we call Policy Controlled Release, which formalizes information-flow correctness according to our notion of declassification policy. We demonstrate, through examples, that our approach works for several interesting and useful declassification policies, including one involving declassification of the average of several confidential values. Finally, we extend the static analyzer to build a practical hybrid static-runtime enforcement mechanism, consisting of 3 steps: static analysis, pre-load checking, and runtime enforcement. We demonstrate how the hybrid mechanism is able to enforce real-world policies which are unable to be treated by standard approaches from industry. Also, we show how this goal is achieved by keeping the static analysis step system independent, and the runtime enforcement with minimum runtime overhead.