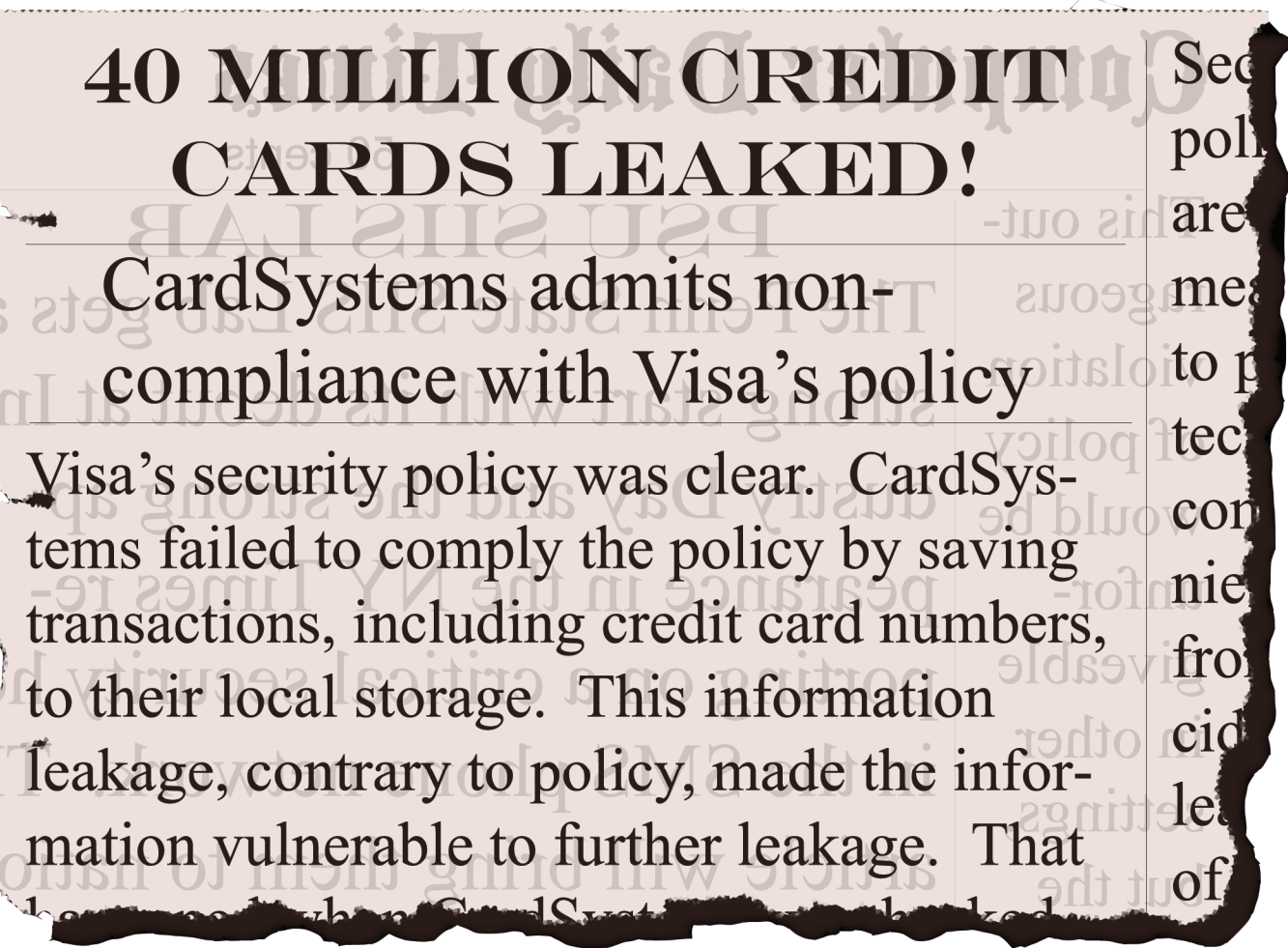
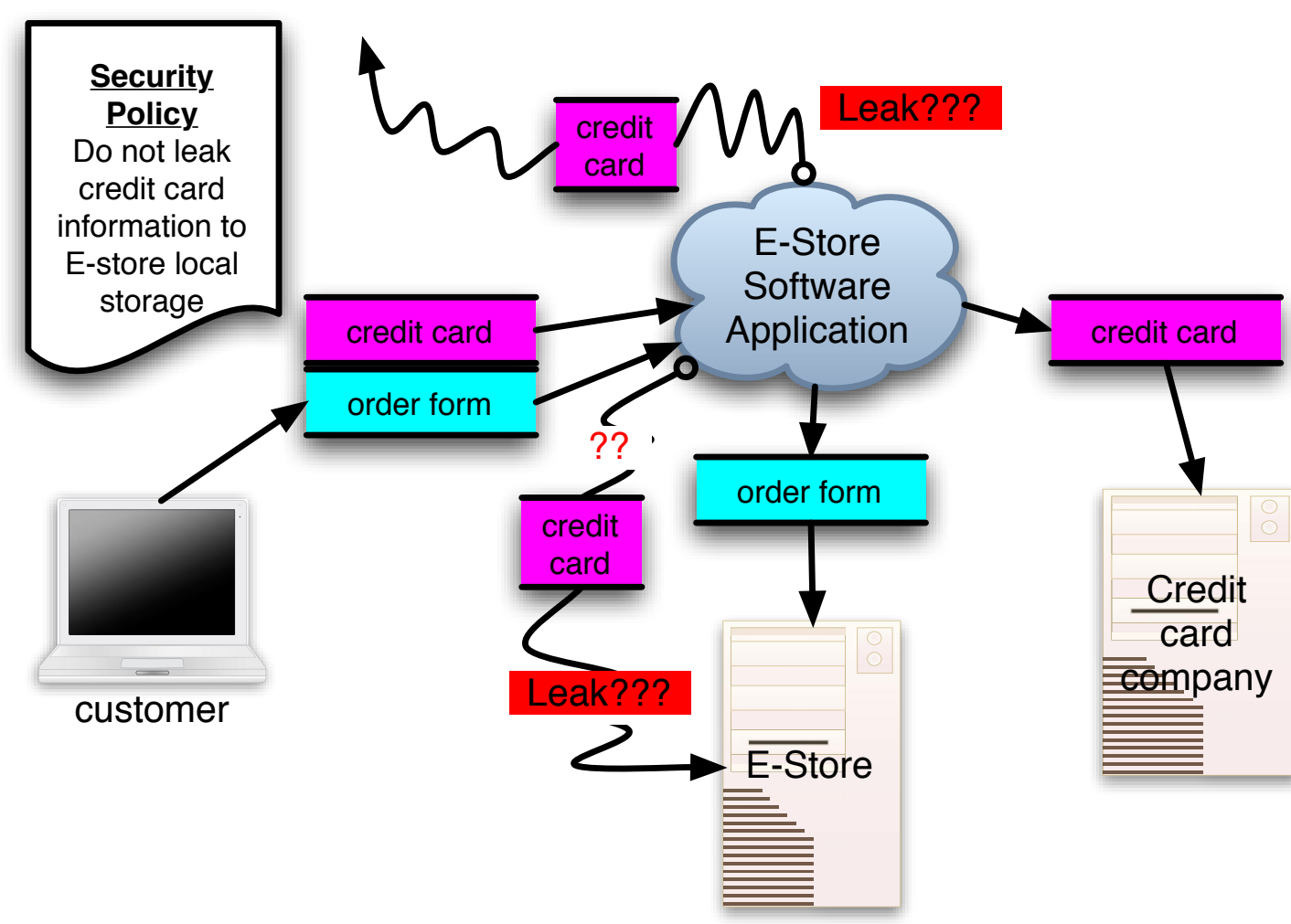


## Policy is Not Enough



## Eli Lilly Releases Confidential Info

A new software application unintentionally emailed the names of 669 Prozac users out to each user on the listserve. Although they maintain a clear privacy policy, they unintentionally violated their own policy...



## Provable Confidentiality Guarantees

Although security policy dictates acceptable information flows, provable guarantees are needed for critical software systems. Even extensive manual certification processes cannot provide enough confidence that software is implemented according to security policy specifications. This was seen clearly in the Diebold electronic voting software which passed certification, but was later discovered to contain serious security flaws.

## End-to-End Information Flow Control

How to prevent information leaks?

**Problem:** private data (like credit card numbers) are handled by applications and can be leaked maliciously or accidentally, contrary to high-level policy.

**Goal:** to ensure the confidentiality of private data (to enforce high-level policy) as it passes from one application to another.

**How:** tag data types with a security level; type-secure programming languages *provably guarantee* noninterference, i.e., that high-secure data and low-secure data don't interact.

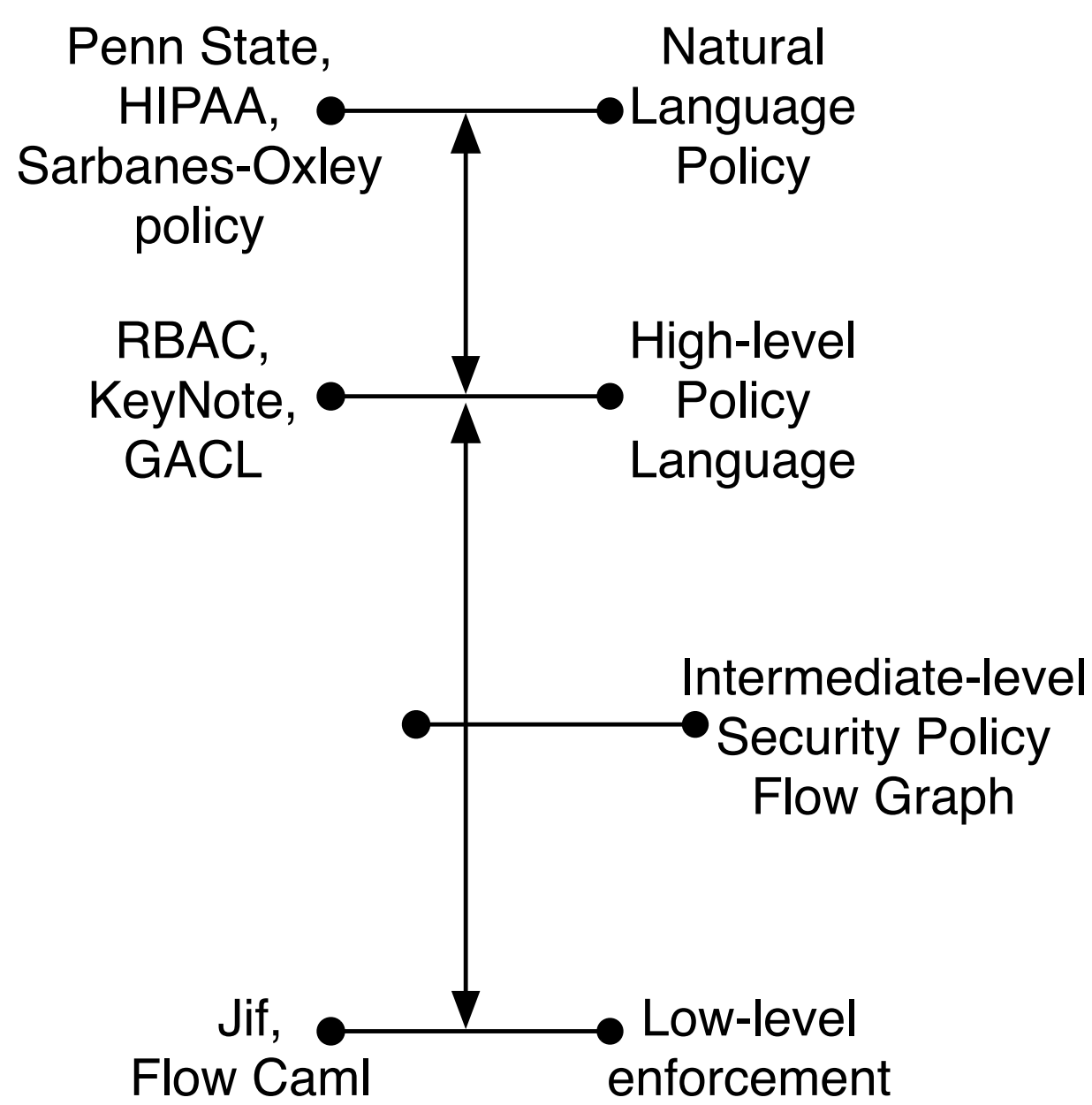
**Tools:** security-typed languages automatically enforce noninterference. By using type analysis, strong security properties can be verified at compile-time.

### Advantages:

1. Using static analysis can enforce stronger properties than the common runtime approach of using reference monitors.
2. Runtime performance is improved by being able to perform the analysis at compile time.
3. The type properties could be used to generate proof-carrying code for verification of the security of untrusted applications.

## From Policy to Enforcement

### Bridging the Semantic Gap



**Natural language policies** such as HIPAA and Sarbanes-Oxley clearly state security and privacy requirements for sensitive information.

**High-level policy languages** such as RBAC, KeyNote and the General Access Control Language (GACL) use mathematical logics and have a clearly defined semantics.

Fine-grained policies may be enforced with *provable guarantees* using **security-typed languages**, like Jif or Flow Caml.

**Problem:** Unfortunately, there is currently no way to write policy in a high-level language and automatically enforce it with a security-typed language.

**Solution:** Develop an *intermediate-level security policy flow graph* which can be used to map high-level policy onto a security-typed language

## Trusted Declassification

We have extended security-typed languages to facilitate a mapping of high-level security policy onto a security-typed application.

**Problem:** Noninterference is too strict of a security policy for real applications. For example, a ciphertext releases information about sensitive inputs, even though it is too little information to recover the inputs. In strict noninterference, this is prohibited. The method for bypassing this stringent security requirement is called *declassification*. Declassification must be applied only in a controlled fashion.

**Solution:** Trusted declassification allows a data owner to indicate which functions are trusted to declassify sensitive data. One example is an encryption function such as AES. Another might be a function which only releases a single bit of information such as in a password-checking program.

## Developing Secure Applications

### A New Approach

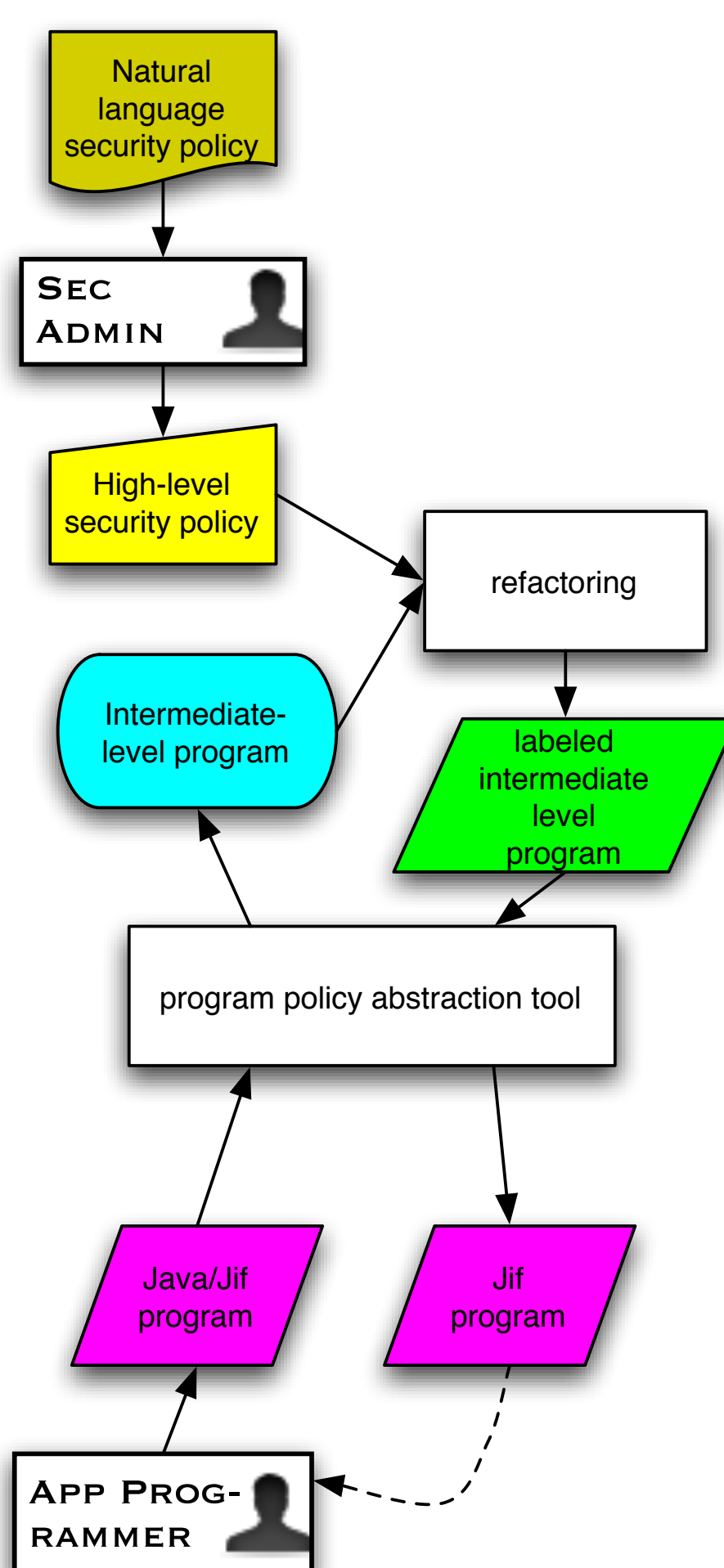
#### Intermediate-level Security Policy Flow Graph

This program and policy structure will be low level enough so that it can be automatically generated from a security-typed program using our *program policy abstraction tool*.

The intermediate-level representation will be high-level enough that it will facilitate an automatic mapping of policy labels from a high-level security policy language.

By using our *refactoring tools*, the intermediate level framework and a high level policy can be combined to generate policy labels automatically for a security-typed program.

This intermediate-level security policy flow graph will bridge the semantic gap and allow for high-level security policies to be implemented efficiently in real applications with provable guarantees of compliance.



### New Developer Tools

#### A Jif IDE using Eclipse

Jif development is prohibitive for large applications, as two recent case studies (including our own) have shown. If Jif programming is to become practical, some tools are needed to aid in application development.

IBM's Eclipse framework has been used for building various Integrated Development Environments, notably for Java and C. We are developing a new IDE for secure programming in Jif.

#### Visualization Tools

1. One of the most challenging parts of Jif programming is visualizing information flows. Color syntax and integrated graphing tools will aid in this.
2. The complexity of Jif annotations can become confusing. The ability to hide and expose annotations simplifies the task of the programmer.

#### Automation Tools

1. The Jif programmer faces both tedious and challenging tasks. Automation tools assist in both. Assisted annotation of native Java methods is an example of reducing tedious tasks.
2. Refactoring tools will provide automated inference of security labels, derived from an intermediate-level security policy flow graph.