AFRL-AFOSR-VA-TR-2019-0357

Policy Enforcement by Using Security Labels

Fred Schneider
CORNELL UNIVERSITY

10/29/2019
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTA2
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
25-10-2019	Final Report	01-06-2019 - 31-05-2019

4. TITLE AND SUBTITLE

Policy Enforcement by Using Security Labels

5a. CONTRACT NUMBER

FA9550-16-1-0250

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Fred B. Schneider

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Computer Science Department, Gates Hall
Cornell University
Ithaca, New York 14853

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Office of Science and Research
875 Randolph Street
Suite 325, Room 3112
Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distributtion A - Approved for Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT

An alternative to associating policy-specific guards with operations is to associate policy-specific security labels with values. A label would specify how the tagged information may and may not be used. This project investigated the feasibility of this alternative approach by developing \emph{reactive information flow} (RIF) labels and

15. SUBJECT TERMS

Computer security, information flow, privacy, reactive information flow

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Fred B Schneider
U	An alternative	U	UU	1	19b. TELEPHONE NUMBER *(Include area code)* 607 255 9221

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE. Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE. State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATE COVERED. Indicate the time during which the work was performed and the report was written, e.g., Jun 1997 - Jun 1998; 1-10 Jun 1996; May - Nov 1998; Nov 1998.

4. TITLE. Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER. Enter all contract numbers as they appear in the report, e.g. F33315-86-C-5169.

5b. GRANT NUMBER. Enter all grant numbers as they appear in the report. e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER. Enter all program element numbers as they appear in the report, e.g. 61101A.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER. Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER. Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/ monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.

Policy Enforcement by Using Security Labels

AFOSR Grant FA9550-16-1-0250

Final Report

01 June 2016 – 31 May 2019

Fred B. Schneider
Computer Science Department
Cornell University
Ithaca, New York
(607) 255-9221 (phone)
fbs@cs.cornell.edu

1 Objectives

An alternative to associating policy-specific guards with operations is to associate policy-specific security labels with values. A label would specify how the tagged information may and may not be used. This project investigated the feasibility of this alternative approach by developing *reactive information flow* (RIF) labels and exploring both static and dynamic means of enforcement of security policies the labels specify.

Specific accomplishments under the auspices of this AFOSR funding include:

- Formulation of a security property appropriate for programs that must not leak classified information but employ reclassification. The new property is called *piecewise noninterference* (PWNI); it generalizes classical noninterference.
- Development of a generic type system for static enforcement of PWNI in programs that use RIF labels to tag values and variables. An instance of this type system was installed and evaluated in JIF, a security-typed programming language.
- Design of dynamic information-flow enforcement schemes based on chains of labels. These schemes allow a label to be associated with

a label on a variable or on another label. Increased permissiveness is obtained when longer label chains are associated with variables, and theorems were proved to characterize the trade-offs.

- The use of RIF labels for specifying and enforcing use-based privacy policies. This work included implementations and analysis of prototype architectures.

2 Discussion of Completed Research

Publications [1] through [7] in §4 give a detailed account of research completed with support from this AFOSR grant. This section gives the context and broad account.

2.1 Reactive Information Flow (RIF) Lables

In Denning's initial work and in much that has followed—both for confidentiality and integrity—the set of restrictions assigned to the output of an operation is the union of the restrictions associated with its inputs. But by ignoring the operator and the values of inputs, that approach can be too conservative. The most general formulation would assign restrictions to the output of an operation $op(x_1, x_2, \ldots, x_n)$ according to operator op, its inputs $x_1, x_2, \ldots, x_n$, and the restrictions specified by labels associated with those inputs. Yet the output of operation $op(x_1, x_2, \ldots, x_n)$ might warrant fewer restrictions, additional restrictions, or an incomparable set of restrictions than are associated with its inputs.

- With an operation that computes the winner of an election, the inputs are votes and the output is the majority. Each input is secret to the principal casting that vote, whereas the output ought to be readable by any principal. So the output should be associated with fewer restrictions than the inputs.

- A conference-management system matches papers to reviewers, where that matching is generated by a non-deterministic computation. The inputs—a list of reviewers and a list of submissions—can be read by the entire program committee, but conflicts of interest dictate that only a subset of the program committee learn which reviewers are assigned to any given paper. So outputs should be associated with more stringent restrictions than inputs.

Reactive information flow (RIF) labels seek to address these limitations by allowing stronger, weaker, or incomparable restrictions to be associated with the output of an operation, where the new restrictions are determined by the operator and the restrictions on inputs. A RIF label is a tag that gives restrictions on the use of a tagged value v and on all values derived from v.

- For confidentiality, the RIF label might specify which principals can read the tagged value or can read values derived from the tagged value.
- For integrity, the RIF label might specify which principals must be trusted in order to trust the tagged value and any values derived from that tagged value.

Notice that RIF labels are describing end-to-end guarantees—they specify current and future allowed uses of specific values, regardless of what variable stores that value or how that value was derived. In contrast, access control policies restrict access to specific information containers, independent of what value the container stores or how the value it stores was derived.

Two classes of RIF labels were explored in depth as part of the research effort: RIF automata (which are based on ordinary finite state automata) and κ-labels (which add a stack-like mechanism in order to handle the context-free character of protocols involving encryption and an inverse decryption operation). The discussion here will focus on RIF automata; publications [5,7] give details for both classes.

A RIF automaton accepts sequences of reclassifiers and, for each automaton state, specifies (i) restrictions on the associated value and (ii) how those restrictions change according to the history of operations involved in deriving that value. Operations of interest to a programmer are identified by *reclassifiers*. Thus, sequences of reclassifiers are abstract descriptions for the series of operations that were applied to values as program execution proceeds. A sequence of reclassifiers together with a RIF automaton can then provide a basis for determining how the confidentiality or integrity of the output of series of operations differs from that of its inputs. The restrictions are those associated with the state of the RIF automaton that results when the automaton reads that sequence of reclassifiers.

Formally, a RIF automaton is a finite-state automaton whose states map to sets of principals and whose transitions are associated with reclassifiers. A RIF automaton λ is thus defined by a 5-tuple $\langle Q, \Sigma, \delta, q_0, \mathit{Prins} \rangle$, where:

- Q is a finite set of automaton states,

- Σ is a finite set of reclassifiers,
- δ is a total, deterministic transition function $Q \times \Sigma \to Q$,
- q_0 is the initial automaton state $q_0 \in Q$, and
- *Prins* is a function from states to sets of principals.

In theory, the number of states in a RIF automaton could be large; in practice, relatively small RIF automata suffice for representing many policies of practical interest. And we have found that changes to the confidentiality or integrity restrictions associated with a value have straightforward descriptions using RIF automata.

– For confidentiality, a reclassifier *triggers* a *declassification* when it causes a transition whose ending state is mapped to a superset of the principals mapped by its starting state. A reclassifier triggers a *classification* when it causes a transition whose ending state is mapped to a subset of the principals mapped by its starting state.

– For integrity, transitioning to a superset of principals triggers a *deprecation* (since a superset must now be trusted) whereas transitioning to a subset triggers an *endorsement* (because only a subset must be trusted).

Practical experience with a proposed programming construct can be invaluable. Building on the Jif dialect of Java, the JRIF (Java with Reactive Information Flow) language extended Java's type system to incorporate RIF automata. JRIF programmers can tag fields, variables, and method signatures with RIF automata, and the JRIF compiler checks whether a program satisfies whatever flow-based restrictions these RIF automata specify. A JRIF label is a pair comprising a RIF automaton for confidentiality and a RIF automaton for integrity. a c-automaton λ_c and an i-automaton λ_i.

JRIF has the ordinary expressions of Java (e.g., constants, variables, etc.). In addition, an ordinary expression $\mathcal{E}$ can be annotated with a reclassifier `f` by writing

$$\texttt{reclassify}(\mathcal{E}, \texttt{f}).$$

Such an *annotated expression* can appear wherever an ordinary Java expression can (e.g., in right-hand side expressions of assignments or in guard expressions of conditional commands). Expressions not explicitly annotated trigger no transition on JRIF labels.

Sometimes a RIF label in a JRIF program only will become known at run time. To accommodate this, JRIF adopts Jif's *dynamic labels*. JRIF dynamic labels may be instantiated as run-time values, stored in variables, and compared dynamically. Since the actual JRIF label that a dynamic label denotes is not known at compile time, the JRIF label checker requires the programmer to provide code that checks for unsafe flows at run time. For example, consider assignment statement

$$\texttt{y} = \texttt{reclassify(x mod 4, f)}$$

where `x` has been declared to have a dynamic label `L1`, and `y` a dynamic label `L2`. This assignment statement is secure only when $\delta(\texttt{L1}, \texttt{f}) \sqsubseteq \texttt{L2}$ holds. In JRIF, programmers can write `T(L1,f)` to represent a dynamic label whose value is $\delta(\texttt{L1}, \texttt{f})$. So, to ensure that $\delta(\texttt{L1}, \texttt{f}) \sqsubseteq \texttt{L2}$ holds when the above assignment statement executes, the JRIF programmer must insert a conditional test as a guard:

$$\texttt{if (T(L1,f)} \sqsubseteq \texttt{L2)} \textbf{ then } \texttt{y} = \texttt{reclassify(x mod 4, f)}$$

- At compile time, constraint $\texttt{T(L1,f)} \sqsubseteq \texttt{L2}$ informs the type system about the necessary relationship between `L1` and `L2`, because the type system may assume $\texttt{T(L1,f)} \sqsubseteq \texttt{L2}$ holds when the "then" clause starts executing.
- At run time, the system constructs the JRIF label that results from an `f` transition on `L1` and checks whether `L2` is at least as restrictive.

This example also illustrates an interesting property of JRIF labels: the same reclassifier may have different effects on different labels. For some instantiations of `L1`, transitioning according to `f` may satisfy relation $\texttt{T(L1,f)} \sqsubseteq \texttt{L2}$, and for other instantiations of `L1`, that transition may not satisfy this relation.

Notice how RIF labels allow program logic to be separated from information flow policies. This makes JRIF programs easier to write and easier to maintain than Jif programs. Suppose, for example, that a programmer decides that some input value—a game player's name—should not be declassified when formerly it was.

- In JRIF, this change to the program involves modifying the JRIF label declaration on any field storing the player's name. The c-automaton of the label would be inspected and edited so that it contains no transitions to automaton states that map to additional principals.

– To accommodate this change in languages that use classic labels, the programmer must not only find and remove all declassification commands that involve the name field explicitly, but she also must remove all declassification commands that involve any expressions to which the game player's name flows. Getting these deletions right is error prone, since the programmer must reason about the flow of information in the code—something the type system was supposed to do.

2.2 Dynamic Enforcement and Label Chains

Dynamic enforcement mechanisms for information flow control employ tags containing labels to represent sensitivity of values that variables store. These labels can be *flow-sensitive*, meaning that they can change when a value with different sensitivity is assigned to the tagged variable during program execution. Sensitive information might influence which assignments execute and, consequently, determine how and when the flow-sensitive label tagging a variable changes. So flow-sensitive labels can depend on sensitive information.

Inspecting or directly observing flow-sensitive labels might itself leak sensitive information. Consider the following program.

$$\textbf{if } m > 0 \textbf{ then } w := h \textbf{ else } w := l \textbf{ end}$$

Suppose w is tagged with a flow-sensitive label, but the other variables are tagged with *fixed* labels, which do not change during execution: l is tagged with fixed label L (i.e., low), m with M (i.e., medium), and h with H (i.e., high), where $\mathsf{L} \sqsubset \mathsf{M} \sqsubset \mathsf{H}$ holds.

- If $m > 0$ holds, then information flows *explicitly* from h to w and *implicitly* from m (in $m > 0$) to w. When the above program terminates, w should be tagged with flow-sensitive label H, because H is at least as restrictive as the label H that tags h and the label M that tags m.

- If $m \not> 0$ holds, then w should be tagged with flow-sensitive label M when the above program terminates, because M is at least as restrictive as the labels that tag l and m.

So, the flow-sensitive label tagging w depends on whether $m > 0$ holds. Information about m, which is sensitive, leaks to observers that can learn that label.

Blocking an execution based on flow-sensitive labels also might leak sensitive information. Consider the above program but extended with two

assignments.

$$\textbf{if } m > 0 \textbf{ then } w := h \textbf{ else } w := l \textbf{ end};$$
$$m := w;\ l := 1$$

- If $m > 0$ holds, then $m := w$ should be blocked to prevent information tagged H and stored in w from flowing to m; assignment $l := 1$ is not reached.
- If $m \not> 0$ holds, then $m := w$ does not need to be blocked (because w stores information tagged M). Assignment $l := 1$ will execute.

Depending on whether $m := w$ is blocked, principals monitoring variable l (which is tagged L) either do or do not observe value 1 being assigned to l. The decision to block $m := w$ depends on the flow-sensitive label of w, which depends on sensitive information $m > 0$. So $m > 0$ is leaked if observers can detect that $m := w$ is blocked.

To prevent such leaks, *metalabels* might be introduced to represent the sensitivity of information encoded in flow-sensitive labels. For example, the metalabel for w in the above program would be M, corresponding to the sensitivity of information encoded in the flow-sensitive label tagging w. Only principals authorized to read information allowed by the metalabel (i.e., M) would be allowed to observe the label of w. The metalabel that tags w would also capture the sensitivity of the decision to execute $m := w$ and reach $l := 1$. To prevent the implicit flow of that information (which is tagged with M) to variable l (tagged L), assignment $l := 1$ must not be executed.

Since metalabels are flow-sensitive, they too could encode sensitive information, which might leak to observers. It is tempting to employ meta-meta labels to prevent those leaks. However, flow-sensitive meta-meta labels might then leak. We seem to need a *label chain* associated with each variable: a label ℓ_1, metalabel ℓ_2, meta-meta label ℓ_3, etc.

Motivated by these observations, this research project investigated dynamic enforcement mechanisms that employ label chains of various lengths. That required formalizing label chains, which led to a surprising insight: unless a label chain $\langle \ldots, \ell_i, \ell_{i+1}, \ldots \rangle$ is *monotonically decreasing*—$\ell_{i+1} \sqsubseteq \ell_i$ for $i \geq 1$—sensitive information can be leaked. Here is why. Consider a variable x having non-monotonically decreasing label chain $\langle \mathsf{L}, \mathsf{H}, \ldots \rangle$, where $\mathsf{L} \sqsubset \mathsf{H}$. Principals assigned label L are authorized to read the value in x. When read access to x succeeds, these principals conclude that the label of x is L. Thus, success in reading x leaks to a principal assigned L information about the

label of x—even though label chain $\langle \mathsf{L}, \mathsf{H}, \ldots \rangle$ defines the sensitivity of that label to be H. Such leaks cannot occur in monotonically decreasing label chains.

Doing this research also required devising a formal definition for mechanisms that prevent illicit information flows by blocking an execution. This led us to the notion of an *enforcer*, which is an interpreter that produces a subset of possible traces, where the subset complies with the policy being enforced. Enforcers defined in this way are able to model a broad class of run-time mechanisms for information flow security. And for research on characterizing the use of label chains, two classes of enforcers proved significant: ∞-*Enf*, which uses infinite label chains, and the k-*Enf* family of enforcers, which use finite label chains of length k.

Ordinary noninterference is not an appropriate security policy for systems that employ run-time mechanisms to enforce information flow policies, but a straightforward generalization is: Block-safe Noninterference (BNI). BNI is a form of noninterference that incorporates observations on variables and tags and that considers all finite traces—normally terminated and blocked. For research into run-time mechanisms that use label chains, it makes sense to consider an extension of BNI. Extension k-BNI stipulates that if two finite traces of the same command agree on initial values whose sensitivity is at most ℓ, then observations involving variables and the first k label chain elements visible to a principal assigned label ℓ should be the same.

The following specific results were then proved for enforcers and label chains. In these results, an enforcer E' is defined to be *at least as permissive as* an enforcer E if, for all executions of each command, E' allows observations involving at least as many identifiers as E.

- Enforcers that k-*Enf* abstracts will enforce k-BNI.
- Longer label chains allow the k-*Enf* family of enforcers to be more permissive, provided elements in a label chain
  - are not redundant—they are not a function of other elements in the same label chain, and
  - capture the real sensitivity of the elements they tag rather than conservatively approximating it.
- Permissiveness that is possible with longer label chains is limited to values having only certain initial information flow restrictions.

2.3 RIF Labels for Use-Based Privacy

Current approaches to privacy in networked information systems—such as notice and consent and the Fair Information Practice Principles (FIPPs)—are poorly suited to modern networked information systems, where information is collected without user awareness, and data sharing and data analysis are pervasive. *Use-based privacy*, which equates privacy with preventing harmful uses, has been proposed as an alternative. This AFOSR grant supported an investigation into the feasibility of expressing and enforcing use-based privacy. The primary reasons for this thrust were:

- A definition of privacy is increasingly important for our nation, including for the intelligence community and law enforcement who benefit from engaging in various forms of surveillance at home and abroad.
- Use-based privacy seems ideally suited to RIF labels and, thus, it provides an opportunity to understand the use and limitations of this class of tags.

The first step in our exploration of how RIF labels could be used to express use-based privacy was to formulate the *Avenance* privacy specification language. Avenance policies give restrictions in terms of who is using the data, the type of use (i.e., which operation accesses the data), and the purpose of that use. The manner in which an Avenance policy's current authorizations change is encoded as a *privacy automaton*, a finite state automata inspired by RIF.

The next step in this research was to evaluate the extent to which the Avenance language might be used to express and enforce use-based privacy, as exemplified in legal regulations and corporate privacy policies. These legal policies are written in human-interpretable language; whether code implements a particular use therefore cannot be automatically verified. A successful use-based privacy regime would instead need to rely on human auditors to inspect code and determine whether it implements a particular use. Such a regime is likely infeasible in a large-scale, open system like the Internet. But it might be feasible in smaller, closed systems—for example, a health care application or a single tech company. So such applications were the focus of the investitations.

Finally, we explored how Avenance policy compliance might be enforced. Industry efforts to verify compliance with legal requirements and contractual obligations today rely on manual review and code audits, which are time consuming and error prone. Avenance facilitated enforcement by separating policy from code. Specifically, experts specify use-based requirements

as Avenance privacy automata. Independently, developers are expected to annotate application code by labeling regions that correspond to types of uses and by identifying locations where sensitive values are used. The code annotations must still be manually reviewed for correctness, but keeping the annotations independent from the policy specification helps modularity and extensibility. We implemented the Avenance language as a C library, and we built an inline monitoring API for run-time checking of compliance with Avenance policies. We have evaluated the run-time overhead of this implementation.

3 Impacts on the Community

- During the period covered by this AFOSR grant, Schneider was a member of the advisory boards for the following companies: Intel Corporation, Ntrepid Corporation, Riskive Technical Advisory Board (chair); and ZeroFox Technical Advisory Board (chair).

- Schneider served on the following other advisory committees: Computer Science and Telecommunications Board, National Academies; Forum on Cyber-Resiliences, National Academies (chair and founder); Naval Studies Board, National Academies.

4 Publications Supported

1. Fred B. Schneider. A computer scientist musing about the DNC hack. Symposium on Cybersecurity and the Changing International Law of Data, American Journal of International Law. Volume 110 (Feb 2017), DOI, doi.org/10.1017/aju.2017.3.

2. Fred B. Schneider. Impediments with Policy Interventions to Foster Cybersecurity Investment. Communications of the ACM, Vol 61, No. 3 (March 2018), pages 36–38.

3. Fred B. Schneider. Putting Trust in Security Engineering. Communications of the ACM, Vol 61, No. 5 (May 2018), pages 37–39.

4. Fred B. Schneider. History and Context for "Defining Liveness". In Annual Review 2018. Distributed Computing Column 72, ACM SIGACT News, Vol 49, Issue 4 (December 2018).

5. Elisavet Kozyri, Owen Arden, Andrew C. Myers, and Fred B. Schneider. JRIF: Reactive Information Flow Control for Java. In Foundations of Security, Protocols, and Equational Reasoning: Essays Dedicated to Catherine A. Meadows. (Joshua Guttman, Carl Landwehr, Jose Meseguer, and Dusko Pavlovic, eds.) Springer International Publishing, 2019, pp 70-88. With Elisavet Kozyri, Owen Arden, Andrew C. Myers.

6. Elisavet Kozyri, Fred B. Schneider, Andrew Bedford, Josee Desharnais, and Nadia Tawbi. Beyond Labels: Permissiveness for Dynamic Information Flow Enforcement. Proceedings of 32nd IEEE Computer Security Foundations Symposium. Hoboken, New Jersey, IEEE Computer Society, 2019.

7. Elisavet Kozyri and Fred B. Schneider. RIF: Reactive Information Flow Labels. Submitted, *Journal of Computer Security*.