Information-Flow Control with Fading Labels

Andrew Bedford
Université Laval, Canada
andrew.bedford.1@ulaval.ca

***Abstract*—Information-flow control mechanisms generally invest the same amount of resources to protect information of varying importance. In this paper, we introduce the concept of *fading labels*. Fading labels are security labels that stop propagating their taint after a fixed amount of uses. Their use allows mechanisms to spend more resources on more important information.**

I. Introduction

Information-flow control mechanisms are mechanisms that enforce information-flow policies (e.g., information from a top secret file should not be sent over the network). This is usually done by associating sensitive information with a label, which is then propagated whenever the information is used; a process called *tainting*.

These mechanisms, be they static [11], dynamic [2] or hybrid [3], generally invest the same amount of resources to protect information of varying importance. However, in systems where ressources are limited such as smartphones and tablets, it may be more appropriate to spend more resources to protect information that is more important (e.g., passwords), and less resources to protect information that is less important (e.g., current location).

For this reason, we introduce in this paper the concept of *fading labels*. They are labels whose taint stops propagating after a fixed amount of uses.

Contributions:

- We introduce the concept of fading labels and a relaxed version of non-interference called depth-limited non-interference in Section II.
- We discuss their advantages, disadvantages and possible variations in Section III.

II. Fading Labels

To illustrate the concept, consider the program of Listing 1.

```
read value from privateFile;
w := 0;
x := value + 1;
x' := value + 2;
y := x mod 3;
z := y * 4;
write z to publicFile
```

Listing 1.

Normally, variable `value` and all of its derivatives (i.e., variables `x`, `x'`, `y`, `z`) would be tainted with `privateFile`'s label. This approach leads to the tainting of increasingly large portions of programs over time; a problem known as *taint creep*.

The more a program is tainted, the more resources will be needed by the mechanism. This is especially true if the mechanism needs to perform additional computations to prevent leaks through covert channels, such as calling a termination oracle to prevent progress leaks [7] or executing dummy operations to prevent timing leaks [1]. Hence, in order to reduce the amount of resources required, we chose to reduce the number of tainted variables. So that the security of sensitive information is not compromised too much, we chose to do so by limiting the *depth* at which a taint is propagated. For example, in Listing 1, variable `z` is derived from `y`, which is derived from `x`, which is derived from `value` (illustrated in Fig. 1). For this reason, relative to `value`, `x` is at depth 1, `y` is at depth 2 where `privateFile`'s label stops being propagated, and `z` is at depth 3. The idea is to parametrize mechanisms so that labels associated to important information are propagated more deeply than those associated to less important information.

For our purposes, we assume that the levels of information are organized in a finite lattice $(\mathcal{L}, \sqsubseteq)$ which contains at least two elements: L for the bottom of the lattice (least important) and H for the top of the lattice (most important), i.e. $\forall l \in \mathcal{L}, L \sqsubseteq l \wedge l \sqsubseteq H$. To each level $l \in \mathcal{L}$ is assigned an integer $\mathit{maxDepth}(l)$ representing the maximum propagation depth. Note that alternatively, the maximum propagation depth can be associated to channels of information rather than their level. So there could be a channel of level H and depth 5, and another one of depth 500.

We can formally define fading labels as sets of couples where the first element is the level of information and the second element is a counter that keeps track of the depth:

$$\textit{(fading labels)} \quad \ell ::= \mathcal{P}(\mathcal{L} \times \mathbb{N})$$

Each time the label is propagated, either due to a data dependence or control dependence, its counters are decremented. Once a counter reaches 0 (e.g., `y` in Fig. 1), then the couple is removed from the set. We use sets because in our context, a variable can be tainted with more than one element of the lattice. For example, if we have an assignment `a := b + c` where `b`:$\{(H, 8)\}$ and `c`:$\{(M,10)\}$, then `a`:$\{(H,7),(M,9)\}$. Note that if `c`:$\{(M,7)\}$, then `a`:$\{(H,7)\}$ because $M \sqsubseteq H$ and the remaining depth of H is greater or equal than M's.

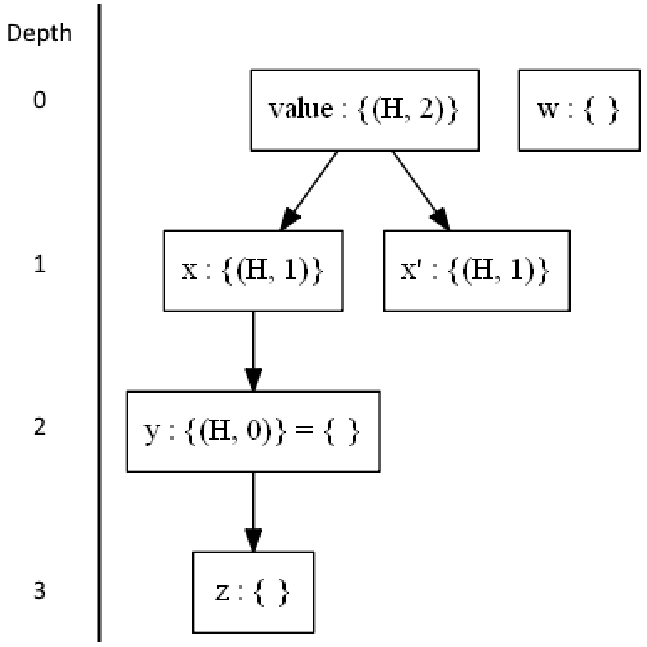

Fig. 1. PDG-like representation of Listing 1

Depth-Limited Non-interference

Non-interference [4] is the security policy that is enforced by most information-flow mechanisms. Intuitively, it states that private information should not interfere with the publicly observable behavior of a program. More formally, it states that there should not be information-flows from inputs of level l_1 to outputs of level l_2 if $l_1 \not\sqsubseteq l_2$. Since fading labels stops propagating after a certain point to reduce resource-usage, mechanisms that use them do not necessarily satisfy non-interference. What they *do* satisfy is a relaxed version of non-interference that we call *depth-limited non-interference.*

In order to define depth-limited non-interference, we use *program dependence graphs* (PDG). They are a visual representation of information flows that can occur in a program. Each node represents a program statement or expression and there are two kinds of edges:

- Data dependence (a.k.a. explicit flows): An edge $x \longrightarrow y$ means that statement x assigns a variable that is used in statement y.
- Control dependence (a.k.a. implicit flows): An edge $x \dashrightarrow y$ means that the execution of y depends of the value of expression x (typically the condition of an if/while command).

If there is a path from node x to node y, it means that information can flow from x to y. So if there are no paths from private inputs to public outputs, then the program is non-interferent. Consequently, PDG-based mechanisms such as Hammer et al. [5] enforce non-interference by searching for such paths, no matter their length. This would reveal that the program in Listing 1 does not satisfy non-interference as there is a path from `value` (private input) to `z` (public output).

Depth-limited non-interference is essentially the same thing, but the verified paths have a maximum length. For example, since the maximum depth of H-level information is set to 2 in Fig. 1, the program would satisfy depth-limited non-interference.

III. Discussion

A. Advantages

The use of fading labels increases the usability of information-flow control mechanisms by lowering the amount of resources needed and by increasing its permissiveness. It provides users with an easy way to parametrize mechanisms so that more resources are used to track important information and less resources are used to track less important information. Furthermore, since fading labels are similar to regular labels, they can easily be integrated into existing mechanisms.

A similar effect could be attained using multiple enforcement mechanisms and regular labels: there could be one mechanism per level of information and their precision could vary in function of this level. However, compared to fading labels, the simultaneous use of multiple enforcement mechanisms would introduce a significant runtime overhead.

Depth-limited non-interference is useful in scenarios where it is too costly to verify that a program satisfies non-interference and where an approximation is sufficient. For example, verifying concurrent programs is costly because every possible interleaving of events has to be considered. Depth-limited non-interference restricts the length of information-flow paths that have to be checked and hence reduces the cost of verification.

B. Disadvantages

While the use of fading labels increases the usability of information-flow control mechanisms, it also reduces their security; leaks of sensitive information may occur. In particular, a malicious application that is aware that it is being monitored by a mechanism which uses fading-labels could circumvent the mechanism and leak sensitive information (e.g., by inserting long dependence paths).

Another disadvantage is that there is no easy way to determine the “right” amount of uses after which a label should stop being propagated; it depends on the application being analyzed and the user’s needs. Static analysis could be used to suggest values that help reduce the overhead introduced by the mechanism, while keeping the number of leaks to a minimum. This could be done by calculating the percentage of input variables that are tracked end-to-end. That is, the percentage of input variables of level ℓ for which there are no paths of length greater than *maxDepth*(ℓ) that lead to an output variable of lower level.

C. Variations

Here are a few interesting variations of the idea.

1) Time-Based Fading Labels: Instead of parametrizing fading labels with taint depths, timespans could be used so that the propagation stops after a certain amount of time. While this would not exactly respect depth-limited non-interference, it could be more intuitive to some users.

2) Usage-Based Fading Labels: Based on the observation that the further a variable is from the original source of sensitive information, the more likely it is that it will have lost information, our proposal in Section II decreases the counter each time the taint is propagated. However, this observation may not always be true. A safer alternative would be to decrease the counter only when non inversible operations are used (e.g., modulo operation). That is, only when we are sure that information is lost. This idea is closely related to the work in quantification [10], which aims at quantifying how much information is leaked by a program or output.

3) Probabilistic Fading Labels: Fading labels as defined in Section II stop propagating their taints once a certain depth has been reached. Another idea would be to parametrize fading labels with probabilities so that low-level variables have a low probability of propagating their taint, and high-level variables have a high probability.

D. Related Work

As far as we know, we are the first to propose a way to vary the amount of resources used by enforcement mechanisms by level of information. That being said, to reduce the amount of resources, fading labels automatically downgrades labels, a process known as *declassification.* Declassification is widely studied in language-based security [9]. It is typically used as a way to safely release sensitive information.

Sabelfeld et al. [8] introduces a notion called delimited release which stipulates that information may only be declassified via **declassify** commands which must be manually inserted into the code. Fading labels on the other hand automatically declassify information.

Kozyri et al. [6] propose to use automatons as labels. The automaton's state determines how the content of a variable can be used. Fading labels could be seen as a specific (and simpler) instance of this.

E. Future Work

We intend to use fading labels in an information-flow mechanism for Android. This will allow us to see how the idea holds in realistic scenarios. More specifically, we will empirically evaluate the performance of mechanisms with and without fading labels.

We also intend to further formalize the approach and develop a static analysis tool that can help users choose the maximum depths.

References

[1] J. Agat, "Transforming out timing leaks," in *POPL 2000, Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Boston, Massachusetts, USA, January 19-21, 2000*, 2000, pp. 40–53. [Online]. Available: http://doi.acm.org/10.1145/325694.325702

[2] T. H. Austin and C. Flanagan, "Efficient purely-dynamic information flow analysis," in *Proceedings of the 2009 Workshop on Programming Languages and Analysis for Security, PLAS 2009, Dublin, Ireland, 15-21 June, 2009*, 2009, pp. 113–124. [Online]. Available: http://doi.acm.org/10.1145/1554339.1554353

[3] A. Bedford, J. Desharnais, T. G. Godonou, and N. Tawbi, "Enforcing information flow by combining static and dynamic analysis," in *Foundations and Practice of Security - 6th International Symposium, FPS 2013, La Rochelle, France, October 21-22, 2013, Revised Selected Papers*, 2013, pp. 83–101. [Online]. Available: https://doi.org/10.1007/978-3-319-05302-8_6

[4] J. A. Goguen and J. Meseguer, "Security policies and security models," in *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*, 1982, pp. 11–20. [Online]. Available: https://doi.org/10.1109/SP.1982.10014

[5] C. Hammer and G. Snelting, "Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs," *Int. J. Inf. Sec.*, vol. 8, no. 6, pp. 399–422, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10207-009-0086-1

[6] E. Kozyri, O. Arden, A. C. Myers, and F. B. Schneider, "Jrif: Reactive information flow control for java," Tech. Rep., 2016. [Online]. Available: https://ecommons.cornell.edu/handle/1813/41194

[7] S. Moore, A. Askarov, and S. Chong, "Precise enforcement of progress-sensitive security," in *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, 2012, pp. 881–893. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382289

[8] A. Sabelfeld and A. C. Myers, "A model for delimited information release," in *ISSS*, ser. Lecture Notes in Computer Science, vol. 3233. Springer, 2003, pp. 174–191.

[9] A. Sabelfeld and D. Sands, "Declassification: Dimensions and principles," *Journal of Computer Security*, vol. 17, no. 5, pp. 517–548, 2009. [Online]. Available: http://dx.doi.org/10.3233/JCS-2009-0352

[10] G. Smith, "Recent developments in quantitative information flow (invited tutorial)," in *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, 2015, pp. 23–31. [Online]. Available: http://dx.doi.org/10.1109/LICS.2015.13

[11] D. M. Volpano, C. E. Irvine, and G. Smith, "A sound type system for secure flow analysis," *Journal of Computer Security*, vol. 4, no. 2/3, pp. 167–188, 1996. [Online]. Available: http://dx.doi.org/10.3233/JCS-1996-42-304