Short Paper: Dynamic leakage – a need for a new quantitative information flow measure

Nataliia Bielova
Université Côte d'Azur, Inria, France
nataliia.bielova@inria.fr

ABSTRACT

A number of measures for quantifying information leakage of a program have been proposed. Most of these measures evaluate a program *as a whole* by quantifying how much information can be leaked *on average* by different program outputs. While these measures perfectly fit for static program analyses, they cannot be used by dynamic analyses since they do not specify what information an attacker learns through observing one concrete program output.

In this paper we study the existing definitions of quantitative information flow. Our goal is to find the definition of *dynamic leakage* – it should evaluate how much information an attacker learns when she observes *one program output.* Surprisingly, we find out that none of the existing definitions provide a suitable measure for dynamic leakage. We hence open a new research question in quantitative information flow area: which definition of dynamic leakage is suitable?

1. INTRODUCTION

Information flow security is an important problem in today's information systems. It enforces limits on dissemination of secret information and is often formulated as a confidentiality property – it requires that an untrusted program should not "leak" secret information into publicly observed outputs. This requirement, formalised as *noninterference*, is often too restrictive in practice since it works in *all-or-nothing* setting: either all or none of information flows from secret to public are allowed.

To achieve a more flexible security requirement, the researchers in quantitative information flow field have proposed a number of measures to evaluate how much information about the secret a program leaks. If the quantity of information is equal to zero, then noninterference holds. Otherwise, the measure provides a number of information bits that evaluates how much information about the secret is being leaked by the public output.

The most popular definitions are based on measuring the decrease in attacker's *uncertainty* about the possible values of the secret and use Shannon entropy [10], Min entropy [20], Guessing entropy and g-leakage [1]. An interesting aspect of the proposed definitions is that all of them measure *a program as a whole.* In other words, they evaluate how much information is leaked *on average* by all possible program outputs. We will call these measures "average measures". To analyse the leakage of a program as a whole, average measures were previously used by a number of works on static program analysis [5, 16] and even approximated by a dynamic analysis [18].

An alternative approach to measure information leakage is to reason about attacker's *accuracy* in his belief about the secret [9]. This belief tracking measure evaluates the leakage *for a concrete secret, and a concrete program output.* Belief tracking has been used in evaluating the worse case leakage through the static program analysis [17].

We graphically show the difference in the definitions of existing measures in Figure 1. We present a program that maps secret input values into output values. The average measures (such as the ones based on Shannon entropy, Min entropy etc.) evaluate how much information from all secret input values flows into all possible output values – we graphically show it by highlighting all secret inputs and all outputs of the program. Belief tracking measure instead only evaluates the flow of information about one secret input into one concrete output – we graphically show it by highlighting only one concrete secret value and one output value.

In the past, the existing measures were used mostly by static program analysis. However, some dynamic programming languages, such as JavaScript, are very hard to analyse statically. Such languages are usually analysed either dynamically [15] through monitoring only one program execution, or through some form of hybrid analysis [14, 19] where non-executed branches can be also analysed. Such analyses have a potential capability to evaluate *how much information is leaked by the current program output.*

In our previous work [6, 7] we have proposed a hybrid analysis that computes the knowledge of the attacker. Our modelling of attacker knowledge is more general than the existing works on dynamic or hybrid analyses that enforce an all-or-nothing requirement of noninterference [22, 3, 11, 4, 8]. Instead, we follow the definition of knowledge from [2] and compute the attacker knowledge as a set of initial secrets that can lead to the currently observed output. We graphically show our modeling in the rightmost part of Figure 1 – the highlighted secret input values that can lead to one

PLAS'16, October 24 2016, Vienna, Austria
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4574-3/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2993600.2993607

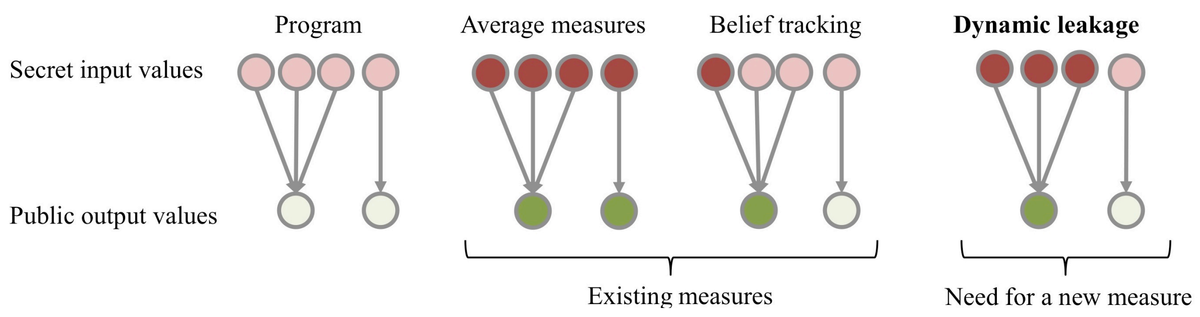

Figure 1: Different measures of information leakage.

concrete output should be analysed in order to measure the dynamic leakage to the attacker. We cannot use the belief tracking measure to measure the leakage associated to one output, because it applies only to a concrete secret input, which is not known to the attacker.

What we think is missing in the field of quantitative information flow is the measure of *dynamic leakage* associated to one program output. Intuitively, such definition evaluates an attacker's knowledge and should not be bounded to a concrete secret since it tries to evaluate the knowledge of the attacker who does not know the secret value.

In this paper, we investigate the current definitions of information leakage and find out that

- for the static case, when the program is evaluated *as a whole*, many existing average measures can be applied, such as Shannon entropy, Min entropy etc.;
- for the dynamic case, when *only one program output* should be evaluated, only the belief tracking definition provides a suitable evaluation of leakage, but only for deterministic programs.

2. BACKGROUND

For simplicity of demonstration, in this work we consider total programs c with just one input S, which is secret, and one output O, which is a publicly observed by an attacker. Thus, we do not consider programs that receive both secret and public inputs, or programs that do not terminate. The attacker provides an untrusted program c, and her goal is to guess the secret S by observing a program output O.

The variable S ranges over some finite set of possible values $\mathcal{S}$ and we assume that the *a priori* probability distribution of S is correct and publicly known to the attacker. We denote the *a priori* distribution of S by π. Similarly, an output variable O ranges over a finite set of values $\mathcal{O}$. The program c may be deterministic or probabilistic.

Information leakage from the secret input S to a publicly observed output O is usually seen as a difference in the *uncertainty* an attacker has about the possible values of S before and after observing the program output O:

information leaked =
initial uncertainty - remaining uncertainty

Following [20], we can model a program c using a matrix whose rows are indexed by $\mathcal{S}$ and whose columns are indexed by $\mathcal{O}$, where the (s, o) entry specifies the conditional probability of a program output o given a secret input s, formally $p(\mathsf{O} = o|\mathsf{S} = s)$. Each row of the matrix has to sum up to 1. In the future notations, we will denote the program c using its matrix over input S and output O by $p_{\mathsf{O}|\mathsf{S}}$, therefore $p_{\mathsf{O}|\mathsf{S}}(o, s) = p(\mathsf{O} = o|\mathsf{S} = s)$.

An *a priori* distribution π on S and a program $p_{\mathsf{O}|\mathsf{S}}$ uniquely determine (i) the distribution of the possible program outputs $p(o) = \sum_s p(s, o) = \sum_s \pi(s) p_{\mathsf{O}|\mathsf{S}}(o, s)$; (ii) *a posteriori* distribution of the secret S given the program output value o, denoted by $p_{\mathsf{S}|o}$:

$$p_{\mathsf{S}|o}(s) = \frac{p(s, o)}{p(o)} = \frac{\pi(s) p_{\mathsf{O}|\mathsf{S}}(o, s)}{p(o)}$$

Intuitively, information leakage should be measured as a difference in the knowledge about the secret an attacker can derive from an *a priori* distribution π (initial uncertainty before observing any program output), and the knowledge she obtains from the *a posteriori* distribution $p_{\mathsf{S}|\mathsf{o}}$ (remaining uncertainty after observing a program output value o).

Most of the proposed definitions in the literature measure how much information is leaked for the whole program, meaning that the knowledge about the a priori distribution is compared against the knowledge an attacker may get from an a posteriori distribution, *averaged among all possible program outputs*. We present the mostly used definitions based on Shannon entropy and Min entropy and demonstrate that they cannot be adapted to measure leakage associated to one program output. We then discuss whether belief tracking measure is suitable for deterministic and probabilistic programs.

3. EXISTING DEFINITIONS FOR THE WHOLE PROGRAM

Shannon entropy. Several definitions have been proposed to measure the amount of information in the given distribution π. The most natural way is given by information theory – the amount of secrecy is computed as an amount of uncertainty an attacker has about the possible value of secret S. Shannon entropy [10] measures such uncertainty[1]:

$$\mathcal{H}(\pi) = -\sum_{s \in \mathcal{S}} \pi(s) \log \pi(s).$$

After an attacker observed some program output value o, the remaining uncertainty is computed by a Shannon entropy of

[1]In all the definitions logarithms are base 2.

a posteriori distribution after observing o:

$$\mathcal{H}(p_{\mathsf{S}|o}) = -\sum_{s\in\mathcal{S}} p_{\mathsf{S}|o}(s)\log p_{\mathsf{S}|o}(s).$$

To evaluate the remaining uncertainty of the whole program, this entropy is *averaged* among all possible program outputs, weighted by their corresponding probabilities[2]:

$$\mathcal{H}(p_{\mathsf{S}|\mathsf{O}}) = \sum_{o\in\mathcal{O}} p(o)\mathcal{H}(p_{\mathsf{S}|o})$$

Finally, information leakage of the program is measured as a difference between the initial and final uncertainty:

$$\mathcal{L}(\pi, p_{\mathsf{S}|\mathsf{O}}) = \mathcal{H}(\pi) - \mathcal{H}(p_{\mathsf{S}|\mathsf{O}})$$

Example 1 (Deterministic program). *Consider the following program:*

if $\mathsf{S} = s_1$ *then* $\mathsf{O} = a$ *else* $\mathsf{O} = b$

with three possible values of secret S *that we denote by* s_1*,* s_2 *and* s_3*. The a priori distribution* π*, and the a posteriori distributions after observing outputs* a *and* b *are:*

	π
s_1	0.875
s_2	0.0625
s_3	0.0625

	$p_{\mathsf{S}\|a}$
s_1	1
s_2	0
s_3	0

	$p_{\mathsf{S}\|b}$
s_1	0
s_2	0.5
s_3	0.5

Notice that after observing an output a, an attacker uniquely identifies the value of the secret, which in this case is s_1. Naturally, since an attacker has learnt the secret value, or in other words, she has no doubts about the value of S, the remaining uncertainty in this case is equal to zero: $\mathcal{H}(p_{\mathsf{S}|a}) = -1\cdot\log 1 = 0$ bit. In case an attacker observes output b, an attacker believes that s_2 and s_3 are equally possible, therefore her uncertainty is measured to be 1 bit: $\mathcal{H}(p_{\mathsf{S}|b}) = -0.5\cdot\log\frac{1}{2} - 0.5\cdot\log\frac{1}{2} = 1$ bit.

These measures are very intuitive since they reflect the uncertainty of the attacker after she observes a concrete program output. However, standard Shannon entropy-based leakage metric averages among all possible outputs, thus computing $\mathcal{H}(p_{\mathsf{S}|\mathsf{O}}) = p(a)\mathcal{H}(p_{\mathsf{S}|a}) + p(b)\mathcal{H}(p_{\mathsf{S}|b}) = 0.125$ bit. This measure takes into account the probability of each output for the program, where $p(a) = 0.875$ and $p(b) = 0.125$ – this is why the final entropy is biased towards a more probable output a, which leaks 0 bits. Since the initial uncertainty is $\mathcal{H}(\pi) = 0.67$ bits[3], the leakage of the program is $\mathcal{L}(\pi, p_{\mathsf{S}|\mathsf{O}}) = 0.67 - 0.125 = 0.54$ bit.

This leakage only computes an *average* leakage for all program executions. We can try to apply the same reasoning and compare the initial and final entropy after one observation, proposing a definition of dynamic leakage[4]:

$$\mathcal{L}^{dynamic}(\pi, p_{\mathsf{S}|o}) = \mathcal{H}(\pi) - \mathcal{H}(p_{\mathsf{S}|o})$$

Applying this definition to our example and output b, we get the negative reduction in uncertainty:

$\mathcal{H}(\pi)$	$\mathcal{H}(p_{\mathsf{S}\|b})$	$\mathcal{L}^{dynamic}(\pi, p_{\mathsf{S}\|b})$
0.67	1	**-0.33**

Since negative leakage is usually interpreted as absence of information, we conclude that this definition is not appropriate: it conveys that an attacker did not learn anything while observing output b, which is not true – he now concludes that secret s_1 is impossible.

Min entropy. Min entropy definition [20] focuses on the *vulnerability* of the secret being guessed by the attacker in one try. Given a distribution π, it is simply a maximum probability in π (the best strategy for the attacker is to pick the most probable secret):

$$\mathcal{V}(\pi) = \max_{s\in\mathcal{S}} \pi(s)$$

A *min-entropy* of π is given by $\mathcal{H}_\infty(\pi) = -\log\mathcal{V}(\pi)$.

After an observation o, an a posteriori distribution $p_{\mathsf{S}|o}$ is also measured using the notion of vulnerability, which evaluates how likely an attacker will guess the secret in one try after she observed o:

$$\mathcal{V}(p_{\mathsf{S}|o}) = \max_{s\in\mathcal{S}} p_{\mathsf{S}|o}(s)$$

Similarly to Shannon entropy, in order to evaluate the leakage of the whole program, the vulnerability is *averaged* for all possible outputs:

$$\mathcal{V}(p_{\mathsf{S}|\mathsf{O}}) = \sum_{o\in\mathcal{O}} p(o)\mathcal{V}(p_{\mathsf{S}|o})$$

The final min-entropy is $\mathcal{H}_\infty(p_{\mathsf{S}|\mathsf{O}}) = -\log\mathcal{V}(p_{\mathsf{S}|\mathsf{O}})$, and the difference between the initial and final min-entropy computes the leakage of the program:

$$\mathcal{L}_\infty(\pi, p_{\mathsf{S}|\mathsf{O}}) = \mathcal{H}_\infty(\pi) - \mathcal{H}_\infty(p_{\mathsf{S}|\mathsf{O}})$$

Coming back to Example 1, the initial vulnerability of the distribution π is $\mathcal{V}(\pi) = 0.875$. After an observation a, an attacker can efficiently guess the secret in one try, therefore $\mathcal{V}(p_{\mathsf{S}|a}) = 1$, while after observing b, she will guess the secret with 50% chance: $\mathcal{V}(p_{\mathsf{S}|b}) = 0.5$.

However, if we follow the definition and average these vulnerabilities, we will get $\mathcal{V}(p_{\mathsf{S}|\mathsf{O}}) = 0.975$ which shows that there is a big chance to guess the secret (we get a bias towards output a because it's much more likely than b). Finally, the leakage is $\mathcal{L}_\infty(\pi, p_{\mathsf{S}|\mathsf{O}}) = 0.1$ bits. Intuitively, the leakage is so small because it was easy to guess the secret before the program runs (vulnerability is 0.875) and also after the program runs (vulnerability is 0.9375).

Again, we can adapt this definition of leakage in order to evaluate what an attacker learns when she observes a concrete output o:

$$\mathcal{L}_\infty^{dynamic}(\pi, p_{\mathsf{S}|o}) = \mathcal{H}_\infty(\pi) - \mathcal{H}_\infty(p_{\mathsf{S}|o})$$

And again, applying it to output b we get a negative leakage:

$\mathcal{V}(\pi)$	$\mathcal{H}_\infty(\pi)$	$\mathcal{V}(p_{\mathsf{S}\|b})$	$\mathcal{H}_\infty(p_{\mathsf{S}\|b})$	$\mathcal{L}_\infty^{dynamic}(\pi, p_{\mathsf{S}\|b})$
0.875	0.19	0.5	1	**-0.81**

An intuition behind min-entropy is that an attacker should always increase his chances to guess the secret in one try after some output observation, however in this example the vulnerability of guessing the secret in one try has dropped

[2]By $p_{\mathsf{S}|\mathsf{O}}$ we denote that the measure takes into account all the possible values of O, while by $p_{\mathsf{S}|o}$ we denote that the measure takes into account one concrete value o of O.
[3]For readability, we round numbers up to 2 decimal places.
[4]The notion "dynamic leakage" is taken from [12], who studied this definition for min entropy.

from 0.9 before any observation to 0.5 after the observation b. This is the reason why the measure becomes negative. We therefore conclude that this definition is not appropriate to measure dynamic leakage of one program output.

Channel capacity. Given a program, and a certain average leakage measure, channel capacity evaluates the maximum leakage over all possible *a priori* distributions. This measure does not require a fixed a priori distribution and hence can be applied in cases when this input distribution is unknown. Channel capacity for Shannon entropy and Min entropy based leakages, $\mathcal{L}$ and $\mathcal{L}_\infty$, was proven to be equal to $-\log|\mathcal{O}|$ in case of deterministic programs [21].

For Example 1, this measure would evaluate the leakage of the whole program being equal to $-\log 2 = 1$ bit since only two output values are possible. This measure was used in the dynamic analysis [18], where another intuitive explanation of channel capacity is given: if the measure counts that an output o leaks k bits, then there should be a coding of o that is not bigger than k bits. In our example with only two possible outputs, the best coding would amount to 1 bit of information, however, as we will see in the next section, the amount of information that an attacker learns from one concrete output can exceed this average measure.

4. EXISTING DEFINITIONS FOR SECRET/OUTPUT PAIR

Belief tracking. An alternative definition of quantitative information flow has been proposed [9]: instead of reasoning about the decrease in uncertainty of the attacker, it suggests to reason about the change in the accuracy of the attacker's belief about the secret.

It was proven [9, Thm. 4] that belief tracking definition of information leakage is always positive for deterministic programs (differently from the proposals on dynamic leakage discussed in the previous section), however this approach has another limitation: it obliges us to choose a concrete secret for which we measure the leakage, otherwise the measure can give us different evaluations for different secrets in case of probabilistic programs.

Given a concrete value of a secret, that we denote by $\dot{s}$, the approach first defines a *reality* probability distribution where the secret value $\dot{s}$ has probability 1, while all the others are zeros. We denote this distribution[5] by $p_{\dot{s}}$.

The definition is based on a notion of *distance* $\mathcal{D}$ between two distributions, and is proposed to be evaluated via *relative entropy* [13] that quantifies the difference in two probability distributions:

$$\mathcal{D}(p \rightarrow p') = \sum_{s\in\mathcal{S}} p'(s) \log \frac{p'(s)}{p(s)}$$

The amount of information flow provided by one program output value o is then defined as

$$\mathcal{L}^{belief}(\pi, p_{\mathsf{S}|o}, p_{\dot{s}}) = \mathcal{D}(\pi \rightarrow p_{\dot{s}}) - \mathcal{D}(p_{\mathsf{S}|o} \rightarrow p_{\dot{s}})$$

Intuitively, given a concrete secret value $\dot{s}$, belief tracking measures how much closer (in terms of distance) an attacker gets to the secret $\dot{s}$ when he observes some output.

We come back to Example 1, when a secret value is s_2 and an attacker observes an output b. Before any observation the chance to guess this secret is defined by $\pi(s_2)$, which is 0.0625 and it corresponds to $\mathcal{D}(\pi \rightarrow p_{s_2}) = \log\frac{1}{0.0625} = 4$ bits of information. After an observation b, *a posteriori* distribution defines that the probability of guessing s_2 becomes $p_{\mathsf{S}|b}(s_2) = 0.5$ and therefore conveys $\mathcal{D}(p_{\mathsf{S}|b} \rightarrow p_{s_2}) = \log\frac{1}{0.5} = 1$ bit. Therefore, the leakage is a difference in these two distances, which is now 3 bits.

[5]This distribution is denoted by $\dot{\sigma}$ in [9] and is called *point mass distribution.*

In the following table we compute the belief tracking leakage for different initial realities:

$p_{\dot{s}}$	$\mathcal{D}(\pi \rightarrow p_{\dot{s}})$	$\mathcal{D}(p_{\mathsf{S}\|b} \rightarrow p_{\dot{s}})$	$\mathcal{L}^{belief}(\pi, p_{\mathsf{S}\|b}, p_{\dot{s}})$
p_{s_1}	0.193	-	-
p_{s_2}	4	1	**3**
p_{s_3}	4	1	**3**

We do not evaluate the leakage for s_1 because this input is impossible for the output b. Notice that the leakage for both realities where s_2 is the secret value and where s_3 is the secret value, are the same. We generalise this in the following theorem[6].

THEOREM 1. *Given an a priori distribution π, a deterministic program $p_{\mathsf{O}|\mathsf{S}}$, and a concrete program output o, the leakage for all possible secrets $\dot{s}$ that may lead to o, is:*

$$\forall \dot{s} \in \mathcal{S}. p_{\mathsf{O}|\mathsf{S}}(o, \dot{s}) = 1 \Rightarrow \mathcal{L}^{belief}(\pi, p_{\mathsf{S}|o}, p_{\dot{s}}) = -\log p(o).$$

Therefore, for deterministic programs, the belief tracking measure provides a reasonable evaluation of leakage: it does not depend on the concrete value of the secret and is positive.

Notice that for a deterministic program from Example 1, the channel capacity measure computes 1 bit of information leakage, however the amount of information an attacker actually learns when she observes an output b can be much bigger in case of our a priori distribution π[7]:

$$\mathcal{L}^{belief}(\pi, p_{\mathsf{S}|b}) = -\log p(o) = 3 \text{ bits}.$$

We now analyse this measure for probabilistic programs.

EXAMPLE 2 (PROBABILISTIC PROGRAM). *Consider the following program, where $c'{}_{\alpha}[]c''$ is a probabilistic choice between commands c' and c'' with probability α:*

```
if S = $s_1$ then O= $a$ $_{0.81}$[] O= $b$
else O= $a$ $_{0.09}$[] O= $b$
```

where the secret variable S *can take only two possible values, s_1 and s_2. An* a priori *distribution π and* a posteriori *distributions $p_{\mathsf{S}|a}$ and $p_{\mathsf{S}|b}$ are:*

π	
s_1	0.25
s_2	0.75

$p_{\mathsf{S}\|a}$	
s_1	0.75
s_2	0.25

$p_{\mathsf{S}\|b}$	
s_1	0.065
s_2	0.935

Notice that when attacker observes an output a, an a posteriori distribution is simply a shuffle of an a priori distribution.

[6]The proof of the theorem can be found in the appendix.

[7]In the notation $\mathcal{L}^{belief}(\pi, p_{\mathsf{S}|b})$ we do not denote a concrete reality anymore since we have proven in Theorem 1 that this leakage is equal for all possible realities in case of deterministic programs.

The beliefs tracking approach provides us with a mean to measure whether attacker's knowledge about a concrete secret has become *more accurate* when an attacker has observed one program output. We therefore *are obliged* to choose a reality against which we want to quantify the leakage.

For Example 2, we analyse the case when an attacker has observed an output a. We show how we computed the leakage for both realities: one where the secret value is s_1, and another one where the secret value is s_2:

$p_{\dot{s}}$	$\mathcal{D}(\pi \rightarrow p_{\dot{s}})$	$\mathcal{D}(p_{\mathsf{S}\vert a} \rightarrow p_{\dot{s}})$	$\mathcal{L}^{belief}(\pi, p_{\mathsf{S}\vert a}, p_{\dot{s}})$
p_{s_1}	2	0.415	**1.58**
p_{s_2}	0.415	2	**-1.58**

Interestingly, depending on the concrete value of the secret, we can have completely opposite results: in one case an attacker gains 1.58 bits in accuracy, and in the other case an attacker looses the same amount of information.

The belief approach provides a framework to reason about each concrete secret and output separately, but it does not provide any evaluation of the attacker knowledge of one program output.

Even more interestingly, if we apply the dynamic leakage definitions based on Shannon entropy for this probabilistic program, we get zero-leakage because the overall distribution of secret values remains the same after the observation a:

$\mathcal{H}(\pi)$	$\mathcal{H}(p_{\mathsf{S}\vert a})$	$\mathcal{L}^{dynamic}(\pi, p_{\mathsf{S}\vert a})$
0.811	0.811	**0**

Min entropy also gives us a zero-leakage since the probability of guessing the secret in one try did not change:

$\mathcal{V}(\pi)$	$\mathcal{H}_\infty(\pi)$	$\mathcal{V}(p_{\mathsf{S}\vert a})$	$\mathcal{H}_\infty(p_{\mathsf{S}\vert a})$	$\mathcal{L}^{dynamic}_\infty(\pi, p_{\mathsf{S}\vert a})$
0.75	0.415	0.75	0.415	**0**

Usually, zero-leakage corresponds to noninterference – that an attacked did not get any gain in reducing uncertainty, however he has definitely learnt something! We therefore conclude that none of the existing definitions is suitable for probabilistic programs.

5. CONCLUSIONS

In this paper we analysed the existing definitions of information leakage and found out that none of them can be used to evaluate the amount of information an attacker learns while observing one program output.

We therefore open a new research question in the field of quantitative information flow: *which definition of dynamic leakage is suitable?*

Acknowledgments

The author would like to thank Deepak Garg for his inputs on the channel capacity definition, and Tamara Rezk for her feedback on the earlier version of the paper. This work has been partially supported by the ANR project AJACS ANR-14-CE28-0008.

6. REFERENCES

[1] M. S. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith. Measuring information leakage using generalized gain functions. In *CSF'12*, pages 265–279, 2012.

[2] A. Askarov and A. Sabelfeld. Gradual release: Unifying declassification, encryption and key release policies. In *S&P'07*, pages 207–221. IEEE, 2007.

[3] T. H. Austin and C. Flanagan. Permissive dynamic information flow analysis. In *PLAS'10*, pages 3:1–3:12. ACM, 2010.

[4] T. H. Austin and C. Flanagan. Multiple facets for dynamic information flow. In *Proc. of the 39th Symposium of Principles of Programming Languages.* ACM, 2012.

[5] M. Backes, B. Köpf, and A. Rybalchenko. Automatic discovery and quantification of information leaks. In *Proc. of the 2009 Symposium on Security and Privacy*, pages 141–153, 2009.

[6] F. Besson, N. Bielova, and T. Jensen. Hybrid information flow monitoring against web tracking. In *IEEE Computer Security Foundations Symposium, CSF'16*, pages 240–254. IEEE, 2013.

[7] F. Besson, N. Bielova, and T. Jensen. Hybrid information flow monitoring of attacker knowledge. In *IEEE Computer Security Foundations Symposium, CSF'16*, pages 225–238. IEEE, 2016.

[8] N. Bielova and T. Rezk. A taxonomy of information flow monitors. In *International Conference on Principles of Security and Trust (POST 2016)*, volume 9635, pages 46–67. Springer, 2016.

[9] M. R. Clarkson, A. C. Myers, and F. B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 17(5):655–701, 2009.

[10] T. M. Cover and J. A. Thomas. *Elements of Information Theory (2. ed.).* Wiley, 2006.

[11] D. Devriese and F. Piessens. Non-interference through secure multi-execution. In *Proc. of the 2010 Symposium on Security and Privacy*, pages 109–124. IEEE, 2010.

[12] B. Espinoza and G. Smith. Min-entropy as a resource. *Inf. Comp.*, 226:57–75, 2013.

[13] Gareth A. Jones and J. Mary Jones. *Information and Coding Theory.* Springer, 2000.

[14] D. Hedin, L. Bello, and A. Sabelfeld. Value-sensitive Hybrid Information Flow Control for a JavaScript-like Language. In *Proc. of the 28th Computer Security Foundations Symposium.* IEEE, 2015.

[15] D. Hedin and A. Sabelfeld. Information-flow security for a core of JavaScript. In *Proc. of the 25th Computer Security Foundations Symposium*, pages 3–18. IEEE, 2012.

[16] B. Köpf and A. Rybalchenko. Approximation and Randomization for Quantitative Information-Flow Analysis. In *CSF'10*, pages 3–14. IEEE, 2010.

[17] P. Mardziel, S. Magill, M. Hicks, and M. Srivatsa. Dynamic Enforcement of Knowledge-based Security Policies. In *CSF'11*, pages 114–128. IEEE, 2011.

[18] S. McCamant and M. D. Ernst. Quantitative information flow as network flow capacity. In *Proc. of the ACM 2008 Conf. on Programming Language*

Design and Implementation, pages 193–205. ACM, 2008.

[19] J. F. Santos, T. Jensen, T. Rezk, and A. Schmitt. Hybrid typing of secure information flow in a javascript-like language. In *Trustworthy Global Computing TGC'15*, pages 63–78, 2015.

[20] G. Smith. On the Foundations of Quantitative Information Flow. In *Foundations of Software Science and Computational Structures*, volume 5504 of *LNCS*, pages 288–302. Springer, 2009.

[21] G. Smith. Quantifying information flow using min-entropy. In *8th International Conference on Quantitative Evaluation of Systems*, pages 159–167, 2011.

[22] S. A. Zdancewic. *Programming languages for information security*. PhD thesis, Cornell University, 2002.

APPENDIX

A. PROOFS

THEOREM 1. *Given an a priori distribution π, a deterministic program $p_{\mathsf{O|S}}$, and a concrete program output o, the leakage for all possible secrets $\dot{s}$ that may lead to o, is:*

$$\forall \dot{s} \in \mathcal{S}. p_{\mathsf{O|S}}(o, \dot{s}) = 1 \Rightarrow \mathcal{L}^{belief}(\pi, p_{\mathsf{S}|o}, p_{\dot{s}}) = -\log p(o).$$

PROOF. Consider an arbitrary secret $\dot{s} \in \mathcal{S}$ that leads to an output o, then according to the definition of belief tracking measure,

$$\mathcal{L}^{belief}(\pi, p_{\mathsf{S}|o}, p_{\dot{s}}) = \mathcal{D}(\pi \rightarrow p_{\dot{s}}) - \mathcal{D}(p_{\mathsf{S}|o} \rightarrow p_{\dot{s}})$$

where the distance between the distributions is

$$\mathcal{D}(\pi \rightarrow p_{\dot{s}}) = \sum_{s \in \mathcal{S}} p_{\dot{s}}(s) \log \frac{p_{\dot{s}}(s)}{\pi(s)}$$

By the definition of reality $p_{\dot{s}}$, there exists only one secret value $\dot{s}$, such that $p_{\dot{s}}(\dot{s}) = 1$ and $\forall s' \neq \dot{s}$, we have $p_{\dot{s}}(s') = 0$. Therefore,

$$\mathcal{D}(\pi \rightarrow p_{\dot{s}}) = 1 \cdot \log \frac{1}{\pi(\dot{s})}$$

Similarly,

$$\mathcal{D}(p_{\mathsf{S}|o} \rightarrow p_{\dot{s}}) = \log \frac{1}{p_{\mathsf{S}|o}(\dot{s})}$$

Then,

$$\mathcal{L}^{belief}(\pi, p_{\mathsf{S}|o}, p_{\dot{s}}) = \log \frac{1}{\pi(\dot{s})} - \log \frac{1}{p_{\mathsf{S}|o}(\dot{s})} = \log \frac{p_{\mathsf{S}|o}(\dot{s})}{\pi(\dot{s})}$$

By the definition of conditional probability,

$$p_{\mathsf{S}|o}(\dot{s}) = \frac{p(\dot{s}, o)}{p(o)} = \frac{\pi(\dot{s}) p_{\mathsf{O|S}}(o, \dot{s})}{p(o)}$$

Since the program $p_{\mathsf{O|S}}$ is deterministic, and $\dot{s}$ may lead to output o, we have $p_{\mathsf{O|S}}(o, \dot{s}) = 1$. Therefore, $p_{\mathsf{S}|o}(\dot{s}) = \frac{\pi(\dot{s})}{p(o)}$ and

$$\mathcal{L}^{belief}(\pi, p_{\mathsf{S}|o}, p_{\dot{s}}) = \log \frac{p_{\mathsf{S}|o}(\dot{s})}{\pi(\dot{s})} = \log \frac{1}{p(o)} = -\log p(o).$$

□