Hybrid Information Flow Monitoring Against Web Tracking

Frédéric Besson, Nataliia Bielova, and Thomas Jensen
INRIA Rennes - Bretagne Atlantique
Rennes, France
Email: name.surname@inria.fr

Abstract—Motivated by the problem of stateless web tracking (fingerprinting), we propose a novel approach to hybrid information flow monitoring by tracking the knowledge about secret variables using logical formulae. This knowledge can then be transformed into a quantified leakage. We propose a generic hybrid information flow monitor parametrized by a static analysis and identify requirements on the static analysis for soundness and relative precision of hybrid monitors. We instantiate the generic monitor with a combined static constant and dependency analysis and prove that this monitor is more precise than the other hybrid monitors instantiated with other types of static analyses. We then present a hierarchy of hybrid monitors including those based on well-known hybrid techniques for information flow control. The whole framework is accompanied by a formalization of the theory in the Coq proof assistant.

I. INTRODUCTION

Web tracking refers to a collection of techniques that allow websites to create profiles of its users. While such profiles might be useful for personalized advertising, it is generally considered a problem which brings user privacy under attack. Whenever a user opens a new webpage, she has no way to know whether she is being tracked and by whom. The recent survey by Mayer and Mitchell [23] classifies the mechanisms that are used to track a user on the web. Web tracking technologies can be roughly divided in two groups: stateful and stateless.

Stateful trackers store information (e.g., cookies) on the user’s computer. Several groups of researchers have reported on the usage of different stateful trackers on popular websites [34], [25], [4] and have found that some third-party analytics services were using these mechanisms in order to recreate the cookies in case they are deleted [33].

On the legal side, the European Union 2009 “anti-cookie” amendment to ePrivacy Directive 2009/136/EC was accepted and several proposals on web tracking were made [19], [18], [27], [9]. As a consequence, many web sites now explain their cookie policy but so far these regulations impose concrete restrictions only on stateful tracking technologies.

The stateless technologies (often called fingerprinting) collect information about the user’s browser and OS properties, and can distinguish the users by these characteristics. The calculation of the amount of identifying information is based on information theory. Eckersley demonstrated by his Panopticlick project [12] that such identification is quite effective. For a simple illustration of fingerprinting consider the code snippet from Figure 1.

A test name = "FireFox" schematically represents a testing of the browser name (that corresponds to the call of navigator.appName browser API). Another test fonts = fontsSet1 schematically represents a check whether the installed fonts on the browser are the same as in some fontsSet1.

1 x := 0;
2 if (name = "FireFox") then
3 x := 1;
4 if (fonts = fontsSet1) then
5 x := 2;
6 output x;

Figure 1: A possible fingerprinting code

Clearly, the information about the browser’s name does not make the browser uniquely identifiable. According to the results of Eckersley [13], the most information about the uniqueness of a browser is stored in the font set and in plugin details. Hence, in case the tracker observes that x = 1, she would just learn the browser name, while if she observes that x = 2, then she would be close to distinguishing this browser from the others. Letting the tracker know only the browser name might not reveal too much information (especially when one uses a popular browser, such as FireFox), and may be even useful in case the behaviour of the application depends on the browser it is running on. However, making the tracker know about all the fonts installed on the browser reveals too much information about it’s user.

In this paper, we propose a protection mechanism against stateless web tracking based on information flow analysis. Information flow analysis has proven useful in web application security in recent years. Both purely dynamic techniques such as no-sensitive upgrade [1] as well as other techniques based on the idea of secure multi-execution [11], have been implemented for
network messages from the server under his control. Special network abilities: he can only send and receive service to websites. A web tracker does not have any located. He promotes the inclusion of these scripts into or more web servers, where the fingerprinting scripts are

A. Threat model

There exist several analyses for quantifying the information flow of a program, all based on an average amount of leaked information (such as Shannon entropy, min-entropy, guessing entropy, etc). However, average evaluation of information is not sufficient for our web tracking problem. Assume for simplicity that the fonts cannot be checked in the program from Figure 1. Then the only two possible outputs are $x = 0$ and $x = 1$. Consider the case when an observer sees an output $x = 1$. How much information is then contained in $x$?

Entropy based approaches yield an average amount of information about the browser name. The actual amount of information in the fact that $x = 1$ is equal to the amount of information in the fact that a browser name is “FireFox”. According to the calculations from Panopticlick [13], it is equal to $-\log_2 P = 2.29$ bits, where $P = 0.29$ is the probability of a browser name being “FireFox”. Then, the output $x = 0$ means that a browser name is not “FireFox”, which gives $-\log_2 (1 - P) = 0.34$ bits. The classical quantitative information flow definitions will only give an average amount of information in $x$. E.g., Shannon entropy based techniques will compute $H(x) = P(-\log_2 P) + (1 - P)(-\log_2 (1 - P)) = 0.74$ bits. This simple example shows that the entropy of a variable is too strong a compression of information about possible outcomes, and that analysis must determine more precisely the knowledge contained in each variable.

A. Threat model

Our web tracker model is based on the gadget attacker [20]. Like a gadget attacker, a web tracker owns one or more web servers, where the fingerprinting scripts are located. He promotes the inclusion of these scripts into the web pages, offering a tracking or an advertisement service to websites. A web tracker does not have any special network abilities: he can only send and receive network messages from the server under his control.

Except for the gadget attacker capabilities, a web tracker has one distinctive property: he owns a database of the browser fingerprints. Therefore, a web tracker is able to compute the probability distributions for the browser properties that have been fingerprinted. These distributions could be also obtained from other sources, such as Panopticlick [12].

Fingerprinting scripts are essentially programs, so within the program analysis realm we assume that a public observer knows the probability distributions of the secret variables, knows the program source code and observes the public outputs of the program.

B. Contributions

- We propose a novel approach to hybrid information flow monitoring based on tagging variables with the knowledge about secrets rather than with security levels.
- We define a generic hybrid monitor, parametrized by a static analysis, and give generic formal results on relation between soundness and precision:
  - we identify a soundness requirement on the static analysis which is sufficient to prove soundness of a generic hybrid monitor;
  - we prove that a more precise static analyses induces a more precise monitor;
  - within our framework, a sound monitor stays sound if its static analysis is replaced by a less precise static analysis
- We instantiate a generic hybrid monitor with a combination of static dependency analysis and constant propagation, and derive three other monitors by weakening the static analyses (including monitors similar in spirit to those of Le Guernic et al. [21], [20]). We then prove that our hybrid monitor is more precise than three other monitors and establish a hierarchy of hybrid monitors, ordered by precision.

The paper is organized as follows. Section II defines the syntax and semantics of a simple programming language, designed for studying fingerprinting of browser features. Section III reviews basic definitions from quantitative information flow and derives a symbolic representation of knowledge. Section IV presents the generic hybrid monitor and Section V proves its correctness, relative to the correctness of the involved static analyses. Section VI defines a precise hybrid monitor based on constant propagation and dependency analysis and Section VII explains how other, simpler monitors can be obtained as instances of the generic monitor. Section VIII compares with related work and Section IX concludes. The correctness of the framework has been proved using the proof assistant Coq. The Coq model and the machine-checked proof of correctness can be found on an accompanying web page [28].
We develop the monitor for a small, imperative programming language slightly modified to focus on fingerprinting of browser features. We assume an identified subset \( \text{Feat} \) of program variables that represents the browser features. Feature variables can be read but not assigned. We restrict the conditionals in if statements to be comparisons of features with variables and values, as these are the only tests that are relevant for the fingerprinting analysis. We will use the following notations:

- \( \text{Var} \) is a set of all program variables;
- \( \text{Feat} \subseteq \text{Var} \) is a set of variables that represent the browser features, ranged over by \( f \);
- \( \text{Val} \) is a set of values, including Boolean, integers and string values;
- \( x \in \text{Var} \setminus \text{Feat} \) ranges over program variables that are not features;
- \( n \) is a constant: \( n \in \text{Val} \) and;
- \( \odot \) is an arbitrary binary operator.

A program \( P \) is a command \( S \) followed by the output of a variable. The language’s syntax is defined in Figure 3.

\[
S ::= \text{skip} \mid x := E \mid \text{if } B \text{ then } S_1 \text{ else } S_2 \mid \text{while } B \text{ do } S_1 \\
B ::= f \times n \mid x \times n \mid f \times x \\
E ::= n \mid f \mid x \mid E_1 \oplus E_2
\]

Figure 3: Language syntax

The semantics is defined in Figure 2 as an evaluation (“big-step”) semantics over a state, which is composed of a command \( S \) to execute, and an environment \( \rho \), that maps the program variables to their values. The semantics is parametrized by a feature context \( C \in \text{Config} \) mapping features to their values which remains unmodified during the evaluation.

### II. Language

We will take as starting point the definitions of quantitative information flow from Köpf and Basin [10] and Smith [31]. Since our programs are deterministic, every program \( S; \text{output } o \) determines a function from a configuration \( C \) to an output \( v \). In our notation, it means that the program has run under the configuration \( C; (S, \rho_0) \downarrow_C \rho \) and produced an output \( v: \rho(o) = v \). Following [10], [31], a program \( S; \text{output } o \) partitions \( \text{Config} \) according to the final value of \( o \).

**Definition 1** (Equivalence Class). Given a program \( S \), a configuration \( C \), an initial environment \( \rho_0 \) and an output variable \( o \), an equivalence class is defined as

\[
\text{Eq}(S, C, \rho_0, o) = \{(S, \rho_0) \downarrow_C \rho \mid \rho(o) = v\}
\]

Once the program \( S \) executes on the configuration \( C \), a tracker can observe that the actual configuration of the user’s browser is one of \( \text{Eq}(S, C, \rho_0, o) \). How much does this equivalence class tell a tracker? If \( \text{Eq}(S, C, \rho_0, o) = \text{Config} \), then the tracker has not learned anything about the actual configuration, hence no information flow has occurred. At the other extreme, if \( \text{Eq}(S, C, \rho_0, o) = \{C\} \), by observing the output \( o \), a tracker uniquely identifies \( C \), which means total leakage of configuration \( C \). All the other cases represent partial leakage\(^3\).

Consider again the program from Figure 1. For the sake of simplicity, we assume that \( \text{name} \) and \( \text{fonts} \) are the only two browser properties we are interested in. Let the user’s browser be “Opera”. In this case \( x = 0 \), and the tracker cannot make a conclusion about the name of the user’s browser. This partial leakage is precisely captured by the equivalence class of configurations with the name being different from “FireFox”.

\(^3\)We shall disregard information leaks related to execution speed and termination.
In order to calculate the number of identifying bits of a browser configuration, Eckersley [13] uses the notion of *self-information* or *surprisal* from information theory. If the probability of a variable \( f \) to have a value \( v \) is \( P(f = v) \), then the surprisal is \( I(f = v) = -\log_2(P(f = v)) \). Eckersley argues that "surprisal can be thought of as an amount of information about the identity of the object that is being fingerprinted".

Following this definition, we define a leakage function \( \text{Leak} : \mathcal{P}(\text{Config}) \rightarrow \mathbb{R}^+ \) for a set of configurations assuming that the probability of every configuration \( P(C) \) is known (for example, from Panopticlick [12]).

**Definition 2.** The leakage of a set of configurations \( A \subseteq \text{Config} \) is defined as follows:

\[
\text{Leak}(A) = -\log_2 \sum_{C \in A} P(C).
\]

The \( \text{Leak} \) function has the following properties:

- \( \text{Leak}(\text{Config}) = 0 \), which corresponds to the case of non-interference.
- For any couple of sets of configurations \( A_1 \) and \( A_2 \):
  
  \[
  \text{If } A_1 \subseteq A_2 \text{ then } \text{Leak}(A_1) \geq \text{Leak}(A_2)
  \]

**B. Symbolic representation of sets of configurations**

We define an abstract domain of configurations, intended to represent a set of configurations, as follows:

\[
\text{Config}^d \ni cg := tt | ff | f \wedge n | cg \wedge cg | cg \vee cg,
\]

We write \( \mathcal{M}(cg) \) for the models of the Boolean formula \( cg \) i.e., the set of configurations that satisfy the Boolean formula \( cg \).

During our analysis we will compute a Boolean formula \( cg \) for every output of the program. For example, for the output \( x = 2 \) of the program from Figure 1, the resulting formula will be

\[(\text{fonts} = \text{fontsSet1}) \wedge (\text{name} = \text{"FireFox"})\]

Similar to the leakage function for set of configurations, we define a leakage function \( \text{Leak}^d : \text{Config}^d \rightarrow \mathbb{R}^+ \) for Boolean formulas representing sets of configurations.

**Definition 3.** The leakage of a Boolean formula \( cg \in \text{Config}^d \) is defined as follows:

\[
\text{Leak}^d(cg) = -\log_2 \sum_{C \in \mathcal{M}(cg)} P(C).
\]

The \( \text{Leak}^d \) function has the following properties:

- \( \text{Leak}^d(tt) = 0 \), which corresponds to the case of non-interference
- For all \( a_1, a_2 \in \text{Config}^d \):
  
  \[
  \text{If } a_1 \Rightarrow a_2 \text{ then } \text{Leak}^d(a_1) \geq \text{Leak}^d(a_2).
  \]

The last property of a \( \text{Leak}^d \) function is particularly important for our quantitative information flow monitors. By weakening the formula we will always get a better approximation of the leakage, since a weaker formula means that the computed leakage is smaller.

**IV. Generic model of hybrid monitors**

In qualitative (“high-low”) information flow control, Le Guernic et al. [21] has shown that a dynamic information flow analysis can be improved by a static analysis of conditional branches that are not being taken. In this section, we generalise those results for quantitative information flow and define a generic hybrid monitor combining static and dynamic analysis. Moreover, we present a static analysis able to dynamically prove the non-interference of programs that were previously out-of-reach of existing hybrid monitors.

**A. Formal definitions**

The monitor will be defined as an operational semantics, parametrized by the configuration \( C \)

\[(S, (\rho, K)) \xrightarrow{\rho'} (K')\]

with a monitoring mechanism for tracking the information flow from the browser features to the output of the program. The new semantic state \( (\rho, K) \) has the following components:

- \( \rho : \text{Var} \rightarrow \text{Val} \) is the environment for program variables.
- \( K : \text{Var} \rightarrow \text{Config}^d \) is an environment of knowledge about features stored in the non-feature variables.

The knowledge is represented by a formula from the abstract domain \( \text{Config}^d \) (see Section 3).

In traditional information flow analysis, variables are tagged with security levels, while our analysis is based on the knowledge environment \( K \), that represents the knowledge every program variable. This knowledge can either flow directly into the variable through an assignment or indirectly by updating the variable inside a conditional that depends on some feature value. Such a knowledge environment is thus a generalization of a simple dependency function between variables, in that it contains additional information about the values of browser features. For example, a knowledge environment \( K \) may contain the following knowledge about the configuration: \( K(x) = (\text{name} = \text{"FireFox"}) \wedge (\text{fonts} = \text{fontsSet1}) \). The initial knowledge environment \( K_0 \) is defined by \( \forall x.K_0(x) = tt \), which means that no variable contains any knowledge about the browser configuration.

The monitor relies on the auxiliary function \( \kappa \) (defined in Figure 4) for approximating the information coming from expressions. The function \( \kappa \) approximates the information obtained from the evaluation of an expression. Evaluating a feature variable \( f \) will give access
to its value and will therefore transmit the information
\( f = C(f) \) where \( C \) is the configuration of the browser.
Accessing a non-feature variable provides the knowledge
present in that variable as defined by the knowledge
environment \( K(x) \).

The evaluation relation \( \Downarrow_C \) defining the big-step
semantics for the generic hybrid monitor parametrized by
a configuration \( C \) is presented in Figure 4. The rules
[skip], [seq], [ifelse] and [whileloop] correspond to
the rules from the standard semantics and are straightforward.
The rule [assign] updates the value environment
with the new value of \( x \). Notice that in traditional
dynamic and hybrid information flow analysis \cite{29}, variable
\( x \) would be assigned a “high” security level in case
it is assigned within the “high” security context. In our
setting, this would mean that the knowledge in variable \( x 
\) should be updated with the knowledge from the security
context. We do not keep track of the security context,
and, as we show in Section VI our monitors are sound and
even more precise than the monitors that keep track
of the security context.

The rule [ifthen] deals with the implicit flow of information
due to conditionals. Assuming that the Boolean
expression \( B \) evaluates to true, the semantics evaluates
\( S_1 \) and statically analyse the non-executed branch \( S_2 \).
The new monitor state \( [B, K, s', s^2]_\rho \) approximates the
knowledge obtained from both branches. We explain this
combination of states in Figure 5 immediately after the
presentation of the static analysis.

\begin{align*}
\text{[skip]} & \quad (\text{skip}, s) \Downarrow_C s \\
\text{[seq]} & \quad (S_1, s) \Downarrow_C s' \quad (S_2, s') \Downarrow_C s'' \\
\text{[ifelse]} & \quad (\neg B \Rightarrow S_1 \Downarrow_C s') \quad (B \Rightarrow S_2 \Downarrow_C s') \\
\text{[whileloop]} & \quad \text{[whileLoop]} (if B then S; while B do S else skip, s) \Downarrow_C s'
\end{align*}

where
\[
\begin{align*}
\kappa(f)_C^\rho &= f = C(f) & \kappa(x)_C^\rho &= K(x) & \kappa(n)_C^\rho &= tt \\
\kappa(e_1 \oplus e_2)_C^\rho &= \kappa(e_1)_C^\rho \land \kappa(e_2)_C^\rho
\end{align*}
\]

Figure 4: Semantics of a generic hybrid monitor.

B. The role of the static analysis

The hybrid monitor is generic because it is
parametrized on a static analysis providing information
about the branches that are not being executed. The
precision of the hybrid monitor can be improved if we
know that the value of a variable, say \( x \), after the non-
executed branch is identical to the value of \( x \) after
the executed branch. The static analysis computes an
abstract environment and the dependencies of a variable
\( x \), which is a set of variables needed to compute \( x \).
The analysis starts from a concrete environment \( \rho \) of
values from the execution and computes an abstract
state \( s^\rho = (\rho^*, D) \), where \( \rho^* \) is an abstract environment
and \( D \) is the dependency information for each variable.
The role of an abstract environment \( \rho^* \) is to detect
variables whose values are identical on both branches.

The results of the static analysis are used in the
[ifthen] rule using the state combination defined in
Figure 6. The auxiliary function \( \delta \) is used for approx-
imating the information coming from conditionals. The
equations defining \( \delta \) state that the comparison \( f \equiv n \)
of a feature variable and a value will provide exactly
that information. Comparing a non-feature variable \( x \)
with a constant will at most provide the information
about feature variables that was present in \( x \). Finally,
the comparison of a feature and a non-feature variable
\( f \equiv x \) will at most transmit the information present in \( x 
\) and the information that \( f \) is equal to the current value
of \( x \), defined in the environment \( \rho \).

The new environment \( \rho' \) is taken from the result of the
executed branch and the new knowledge environment
\( K'' \) is updated as follows.

If the values of a variable \( x \) are not the same after
the execution of both branches, then \( x \) definitely obtains
a complete knowledge about the conditional \( B \). We
represent this as a conjunction of the knowledge in \( x \)
\( (K'(x)) \) with the knowledge in \( B \) \( (\delta(B)_\rho^\rho) \).

If the values of a variable \( x \) are the same after the
execution of both branches, then the variable \( x \) does not
contain a complete knowledge about the conditional
test \( B \). Instead, from the attacker point of view, the new
knowledge in \( x \) can be obtained either from the executed
branch or the non-executed branch. The formula we ob-
tain can be understood as an abstraction of the standard
\textit{weakest precondition} of a conditional statement:

\[
wp(\text{if } B \text{ then } S_1 \text{ else } S_2) = \left( \neg B \implies wp(S_2) \right) \land \left( B \implies wp(S_1) \right)
\]
Here, the knowledge in $x$ flowing from the non-executed branch is obtained from the knowledge of the variables used to compute $x$ ($\bigwedge_{y \in D(x)} K(y)$) and the knowledge in $x$ flowing from the executed branch is obtained by the monitoring mechanism. Notice that $\delta(B)$ is never the negation of $\delta(B)$ but an abstraction. It is because $\delta(B)$ is by construction an over-approximation of the knowledge of $B$.

Consider the following program:
\[
\begin{align*}
&x := 1; y := 0; \\
&\text{if } (f = 0) \text{ then } y := 1 \\
&\text{else skip;} \\
&\text{if } (g = 0) \text{ then skip} \\
&\text{else } x := y
\end{align*}
\]

Here, $f$ and $g$ are secret variables that are equal to zero in the current configuration. Before the execution of the test $g = 0$, the variable $y$ already contains some knowledge: $K(y) = (f = 0)$. Let’s assume that a static analysis tracks the values and is able to detect that $x$ depends on $y$. The resulting state of the static analysis after evaluating $x := y$ is $\rho'(x) = 1, D(x) = \{y\}$. The resulting state after the execution of the skip branch would remain unchanged. Now, since the value of $x$ would be the same and equal to 1 after the execution of either of the branches, the tracker would conclude that either $g = 0$ or $f = 0$. Our combination of states computes exactly this knowledge: $(\delta(g = 0)_\rho^K \lor K(y)) \land K'(x) = (g = 0) \lor (f = 0)$.

Notice that there is no static analysis involved in purely dynamic monitoring, and still we can model it as a special case of our hybrid monitor. The abstract environment can be seen as $\forall x. \rho'(x) = \top$, and hence we obtain a simple dynamic monitor that does not reason about non-executed branches, but instead pessimistically decides that all the variables will contain knowledge from the tests of the if-statements. This new knowledge in $x$ will then contain the knowledge from the executed branch and from the test $B$: $\delta(B)_\rho^K \land K'(x)$.

V. GENERIC SOUNDNESS AND PRECISION THEOREMS

In this section, we establish the soundness and precision theorems that hold for the generic model of hybrid monitors presented in Section IV. Comprehensive proofs can be found in the companion Coq development [28].

A. Monitor soundness and precision

A concrete hybrid monitor is obtained by instantiating the generic model by a given static analysis, say $A$. In the following, we write $\psi^A$ for an hybrid monitor that uses the static analysis $A$. A monitor $\psi^A$ is sound if after monitoring a statement $S$ it over-approximates the knowledge about features contained in the output variable $x$. The formal statement of this property is given in Definition 4.

Definition 4 (Monitor soundness). A hybrid monitor $\psi^A$ is sound if starting from an initial configuration $C$ and the initial environment $(\rho, \lambda x. tt)$, it monitors a statement $S$ and reaches a final configuration $(\rho', K)$ such that for all variable $x$, $K(x)$ under-approximates the set of undistinguishable configurations

\[\mathcal{M}(K(x)) \subseteq Eq(S, C, \rho, x)\]

The most precise monitor would compute $Eq(S, C, \rho, x)$ that is exactly the set of configurations indistinguishable from $C$ by observing the value of $x$. In general, the closer the set $\mathcal{M}(K(x))$ is to $Eq(S, C, \rho, x)$, the more precise is the monitor.

Definition 5 (Monitor precision). A hybrid monitor $\psi^A$ is more precise than a hybrid monitor $\psi^B$ if for every statement $S$ and initial configuration $C$, the monitor $\psi^A$ always computes a bigger set of configurations corresponding to the knowledge stored in output variable $x$. Formally,

\[
\begin{align*}
&\{(S, (\rho, K_0)) \psi^A_C (\rho_A, K_A)\} \\
&\{(S, (\rho, K_0)) \psi^B_C (\rho_B, K_B)\} \\
&\Rightarrow \\
&\mathcal{M}(K_B(x)) \subseteq \mathcal{M}(K_A(x)).
\end{align*}
\]

This is coherent with the definition of leakage in Section III because the leakage function is anti-monotonic in the set of configurations. Thus, a more precise monitor would estimate a smaller leakage:

\[\text{Leak}^A(K_A(x)) \leq \text{Leak}^B(K_B(x)).\]
In Section VI we will define a static analysis that will induce a precise monitor, and prove that this precise monitor is sound. This has a consequence that we can prove soundness of other monitors by proving that they are less precise than our hybrid monitor. This result is particularly useful when monitors are obtained by weakening the static analysis they employ, as is done when defining the hierarchy of monitors in Section VII.

B. Soundness Requirements for Static Analyzes

The generic hybrid monitor has a generic soundness proof relying only on a requirement for the static analysis. As explained in Section IV the role of the static analysis is to extract executions within the non-executed branch that are indistinguishable from the executed branch and estimate the knowledge that is carried by the variables. Definition 6 provides the formal specification for static analyses that are compliant with our generic hybrid monitor.

**Definition 6 (Sound Static Analysis).** A static analysis \( \downarrow^* \) is sound (for our hybrid dynamic monitor) if the following implication holds:

\[
\begin{align*}
(S, \rho) &\downarrow^C (\rho') \\
(S, \rho_0) &\downarrow^D (\rho^A, D) \\
\rho^A(x) &= v \\
\forall y, y \in D(x) &\Rightarrow \rho(y) = \rho_0(y)
\end{align*}
\]

**Theorem 1 (Soundness).** Suppose a sound static analysis \( \downarrow^* \) according to Definition 6. Then the hybrid monitor \( \downarrow^\sharp \) is sound according to Definition 7 and therefore safely approximates information leakage.

**Proof:** Theorem 1 is obtained directly from the following Lemma, by setting \( S \) to skip. ■

**Lemma 1.** Under the following conditions

\[
\begin{align*}
(S, \rho_0) &\downarrow_C \rho, \\
(S', \rho, K) &\downarrow_C (\rho', K'), \\
\forall x, M(K(x)) &\subseteq Eq(S, C, \rho_0, x)
\end{align*}
\]

we have that, for all \( x \),

\[
M(K(x)) \subseteq Eq(S; S', C, \rho_0, x)
\]

**Proof:** The proof is by induction over the definition of the monitor semantics \( \downarrow \) for \( S' \).

- The case for [skip] is immediate.
- For [assign] \([x' := E]\), if \( x' \neq x \), the value of \( x \) is unmodified and the proof therefore follows directly by induction hypothesis. If \( x = x' \), the proof is by induction over \( E \) and definition of \( k \).
- For [seq], the proof is by induction hypothesis using the associativity of the sequence.
- For if/then, we consider the following cases:
  - If the condition evaluates to tt for \( C' \), the proof follows by induction hypothesis.
  - If the condition evaluates to ff for \( C' \) and the static analysis does not predict the value of the non-executed branch \((\rho^A(x) \neq \rho(x))\), the definition of \( \delta \) forbids the two conditions to evaluate differently and the proof is completed by contradiction.
  - If the condition evaluates to ff for \( C' \) and the static analysis does predict the value of the non-executed branch \((\rho^A(x) = \rho(x))\), we have two cases. If we have \( \delta(B)^K \), the proof follows for the same reason as above. If we have \( \bigwedge_{y \in D(x)} K(y) \), the proof follows using the fact that the static analysis \( \downarrow^\sharp \) is sound according to Definition 6.

- For [whileLoop] and [if/else], the proof follows directly by induction hypothesis. ■

C. Precision Requirements for Static Analyzes

The relative precision of different monitoring mechanisms is often difficult to establish, at least formally. In our generic hybrid monitor the precision of the monitor is directly linked to the strength of the static analysis: a better static analysis yields a more precise monitor.

**Definition 7 (More Precise Analysis).** An analysis \( A \) is more precise than an analysis \( B (A \subseteq B) \) if for any result of the static analysis \( B \) there exists a more precise result output by analysis \( A \) i.e., the abstract environment is more defined and the set of variables computed is smaller.

\[
(S, \rho) \downarrow^\sharp_B (\rho^B, D_B) \Rightarrow \\
\exists \rho_A^B, D_A^B, \bigwedge
\begin{cases}
(S, \rho) \downarrow^\sharp_A (\rho^A, D_A) \\
\forall x, \rho^A(x) = v \Rightarrow \rho^A(x) = v \\
\forall x, D_A(x) \subseteq D_B(x)
\end{cases}
\]

Using the previous definition of precision, we are able to state the following generic theorem.

**Theorem 2 (Relative Precision).** If a static analysis \( A \) is more precise than a static analysis \( B \) (according to Definition 7) then the hybrid monitor \( \downarrow^A \) is more precise then the hybrid monitor \( \downarrow^B \) (according to Definition 5).

The proof is by induction over the definition of the monitor semantics \( \downarrow \) and follows from the fact that all the rules are monotonic with respect to ordering of the knowledge \( K \). This is especially the case for the if/then because, as shown by Figure 3, a stronger analysis computes less spurious dependencies and therefore a weaker formula. Remember that weaker is better and that non-interference corresponds to computing the formula \( tt \). The full proof is part of the Coq development [28].

This theorem is the key for comparing the different existing and novel hybrid monitors presented in Section VI and Section VII.
D. Where are the security contexts?

Security contexts are a traditional ingredient of static and dynamic information flow mechanisms. Perhaps surprisingly, our generic hybrid monitor is sound even in the absence of security context and ignoring the security context leads to a more precise monitor. Our generic hybrid monitor could incorporate a security context \( \sigma \) by rewriting the \([\text{ASSIGN}]\) rule and the \([\text{IFTHEN}]\) as

\[
D' = D[x \mapsto \kappa(E)_C^D \land \sigma] \\
(\text{if } C \text{ then } x := E, (\rho, D, \sigma) \Downarrow_C (\rho[x \mapsto \llbracket E \rrbracket_C^D], D', \sigma)) \\
\llbracket [B]_C^D \llbracket \Downarrow_C \sigma' = \sigma \land \delta(B)_\rho^K (S_1, (\rho, D), \sigma') \Downarrow_C s' \land (S_2, \rho) \Downarrow_C s''
\]

The explanation for this apparent paradox is that our \([\text{IFTHEN}]\) incorporates the knowledge of the current condition and therefore includes the security context on a “lazy” basis.

**Theorem 3** (Security Context). For a given (sound) static analysis, a monitor not using security contexts is always sound and more precise than a monitor using security contexts.

This result is a direct consequence of Theorem 1, and the fact that the assignment rule with security context computes a stronger formula. The proof is also part of the Coq development [28].

It is worth noting that ignoring the security context does not improve the purely dynamic monitor: the security context will eventually be included. However, the improvement is visible for hybrid monitors and allows to prove the absence of information flow in programs like

\[
\text{if } C \text{ then } x := 1 \text{ else } x := 1; \text{ output } x.
\]

VI. A HYBRID MONITOR WITH CONSTANT PROPAGATION AND DEPENDENCY ANALYSIS

In this section we define a hybrid monitor that employs a static analysis which can take full advantage of the concrete values available to the dynamic part of the hybrid monitor. Our static analysis is a combination of constant propagation and dependency analysis. As explained in Section IV-B, the hybrid monitor can take advantage of the fact that a variable has the same value on both branches of a conditional to make a more accurate estimation of the knowledge about features contained in that variable.

An abstract state \((\rho, F) \in \text{State}^\ell\) is a pair of:

- an abstract environment \(\rho : \text{Var} \rightarrow \text{Val}^\ell\) where \(\text{Val}^\ell = \text{Val} \cup \{\top\}\) and \(\top\) represents an arbitrary value,
- a dependency function \(D : \text{Var} \rightarrow \mathcal{P}(\text{Var})\) such that the computation of \(x\) depends upon a set of variables \(D(x)\).

Abstract states are equipped with a partial order \(\sqsubseteq\) obtained as the cartesian product of the ordering of abstract values: \(\forall x, y. x \sqsubseteq y \text{ iff } x = y \lor y = \top\) and the point-wise lifting of the standard set inclusion \(\mathcal{P}(\text{Var})\). The join operator \(\sqcup\) is the least upper bound induced by the ordering \(\sqsubseteq\).

The static analysis of a program \(S\) is defined as a relation between abstract states, written as the judgment \((S, s) \Downarrow s',\) with the intended meaning that \(s'\) is a valid abstraction of the result when running program \(S\) in an initial state that is modeled by abstract state \(s\). The static analysis is specified in Figure 6 as a syntax-directed set of inference rules that generate constraints over abstract states.

The \([\text{AIFCOMB}]\) rule combines the states after the analysis of two branches in case the test of the if-statement can be evaluated in the given abstract environment. The state combination \(\llbracket [B, D, s_1, s_2]_\rho^K\) from Figure 7 is the abstraction of the combination of the states from executed and non-executed branches that we defined in Figure 3. In disjunctive normal form, the logical formula for obtaining \(K''\) is of the form

\[
(\delta(B)_\rho^K \land K'(x)) \lor (\bigwedge_{y \in D(x)} K(y) \land K'(x)) \lor \ldots
\]

In the rest of this section, we explain why the set \(D''\) (see Figure 4) represents an under-approximation of this formula. Note that the static analysis can safely ignore the other terms of the formula, here represented by \(\ldots\). If the values of \(x\) are different after both branches, we combine the knowledge obtained by \(x\) in the executed branch and in the test \(B\). If the values of \(x\) are the same after both branches, then \(x\) gets the knowledge either from the executed branch and the test \(B\) or just from both branches.

The state combination for static analysis in Figure 7 uses auxiliary sets of variables: \(D_{\text{true}}(x)\) and \(D_{\text{both}}(x)\). The set \(D_{\text{true}}(x)\) is the set of variables in the test \(B\) and the set of variables, on which \(x\) depends after the potential execution of the true branch. This set represents the same idea that was used in the state combination of hybrid monitor: it corresponds to the knowledge in the formula \(\delta(B)_\rho^K \land K'(x)\). The set \(D_{\text{both}}(x)\) computes a set of variables on which computation of \(x\) depends in both branches. This set corresponds to the knowledge in the formula \(\bigwedge_{y \in D(x)} K(y) \land K'(x)\).

Now, when we construct a new dependency set \(D''(x)\), in case the values of \(x\) are different the \(D_{\text{true}}(x)\) set is taken. This case is a straightforward translation of the same condition in Figure 5. In case the values of \(x\) are different, we would like to approximate the knowledge we
we cannot approximate the disjunction with the set of variables. Hence, the leakage computed from a smaller set of variables. Therefore, the knowledge about the value of $x$ does not influence the decision of the new dependency set because $D_{true}(x) \subseteq D_{both}(x)$. Hence, $D''(x) = D_{true}(x) = \{y\}$. 

Then, the hybrid monitor combines the results of the static analysis and of the executed branch in the [IFTHEN] rule from Figure 4. In this rule, $D(x) = \{y\}$. Since our monitor does not track the security context, the knowledge in $x$ after the execution of the branch skip is $K'(x) = tt$ and $y$ does not contain any knowledge: $K(y) = tt$. Therefore, $K''(x) = (\delta(B)^K \lor K(y)) \land (\delta(B)^K \lor K'(x)) = tt$.

This example clearly shows that the execution of this non-interferent program will be accepted by our hybrid monitor to compute a conjunction and a disjunction of sets of variables. Hence, the leakage computed from a smaller set of variables. Hence, we propose to choose one set, either $D_{true}$ or $D_{both}$, which is more precise than the other.

Notice, that if $X \subseteq X'$, then the formula for set $X$ that is computed by hybrid monitor, is weaker than the formula for set $X'$, because it is a conjunction of a smaller set of variables. Hence, the leakage computed from $X$ is smaller than the leakage computed form $X'$.

We prove the soundness requirement for the static analysis presented in Figure 6.

**Theorem 4 (Static Analysis Soundness).** The static analysis $\downarrow^2$ is sound according to Definition 6.

Consider a non-interferent program 4 from Table I when B is true, $x = 1$ and otherwise $x = 1$ because $y = 1$. The knowledge about the value of $x$ is contained in $z$, however it does not influence the value of $x$ because there is no execution where $x$ would be assigned to $z$. To explain the static analysis, let’s consider the case when $B$ is true. The static analysis starts from the branch if $(y = 1)$ then $x := y$; else $x := z$ and since the test $y = 1$ can be evaluated, the rule [IFCOMB] is applied. The resulting states from the branches $x := y$ and $x := z$ are combined according to the static analysis state combination, where the auxiliary sets are: $D_{true}(x) = \{y\}$ and $D_{both}(x) = \{y, z\}$, hence $D''(x) = \{y\}$. Here, the value of $z$ does not influence the decision of the new dependency set because $D_{true}(x) \subset D_{both}(x)$. Hence, $D''(x) = D_{true}(x) = \{y\}$. 

Figure 6: Constant propagation and dependency analysis.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ASkip]</td>
<td>${\text{skip}, s} \downarrow^2 s$</td>
</tr>
<tr>
<td>[ASeq]</td>
<td>$(s_1, s_1) \downarrow^2 s_2 \quad (s_2, s_2) \downarrow^2 s_3 \quad (s_1; s_2, s_1) \downarrow^2 s_3$</td>
</tr>
<tr>
<td>[AIfTop]</td>
<td>$|B|_\rho = \top \quad (s_1, s) \downarrow^{i} s_1 \quad (s_2, s) \downarrow^{i} s_2$</td>
</tr>
<tr>
<td>[AIfElse]</td>
<td>$|B|_\rho = \bot \quad (s_1, s) \downarrow^{i} s_1 \quad (s_2, s) \downarrow^{i} s_2$</td>
</tr>
<tr>
<td>[AAssignVal]</td>
<td>$|x := E, (\rho, D)|_\rho = (\rho[x \mapsto \lfloor E \rceil^\rho], D')$</td>
</tr>
</tbody>
</table>

Figure 7: State combination for the [AIFCOMB] rule of the static analysis from Figure 6.
monitor, however other dynamic and hybrid information flow techniques would mark $x$ with "high" security label, since it has been assigned under the security context of a secret condition $B$.

VII. A Hierarchy of Hybrid Monitors

Next, we examine three variants of the monitor from the previous section, obtained by modifying the constant propagation and dependency analyses. These modifications are defined by replacing the rules for assignment and conditionals in the definition of the static analyses (Figure 6). We shall name each monitor by HM(X + Y) where X is the name of the rule for assignment and Y is the rule for conditionals used. The systematic way in which these monitors are derived makes it easy to organize them into a hierarchy of relative precision, depicted in Figure 8. All the precision theorems in this section are a direct consequence of Theorem 2 stating that more precise static analysis induces more precise hybrid monitoring. The proofs of the theorems have been left out for lack of space—see [28].

Table I lists examples of non-interferent programs that illustrate the difference in precision of hybrid monitors. A program is marked as "accepted" when the monitor computes that output variable $A$ does not contain any knowledge about the secrets and "rejected" otherwise.

A. The [HM(VAL + SIMP)] monitor

The precise treatment of conditionals in the static analysis from Figure 6 attempts to determine the actual value of the Boolean conditional by the constant propagation analysis. A simpler analysis would abandon this idea and just assume that both branches might be executed. Instead of [AIfTop], [AIfComb] and [AIfSimple] rules, this analysis uses one simple rule for if-statements:

$$\text{[AIfSimple]} \quad \frac{(S_1, s \downarrow^2 s_1) \quad (S_2, s \downarrow^2 s_2)}{(\text{if } B \text{ then } S_1 \text{ else } S_2, s \downarrow^4 s_1 \sqcup s_2)}$$

Theorem 5. HM(VAL + COMB) monitor is more precise than HM(VAL + SIMP) monitor.

To illustrate the difference in precision, consider a program 4 from Table I that is "accepted" by HM(VAL + COMB) monitor as we showed in previous section. Consider the case when $B$ is true. Either $A$ is true or false, $z$ contains knowledge about $A$. The static analysis of HM(VAL + SIMP) monitor ignores the Boolean conditional ($y = 1$), and hence it computes a set of variables $D(x) = \{y, z\}$. If $A$ is true, then the new environment $\rho'(x) = 1$, and [IfThen] rule of the hybrid monitor computes $K''(x) = (\delta(B)^K_\rho \lor (tt \land K(z))) \lor (\delta(B)^K_\rho \land tt) = (\delta(B)^K_\rho \land K(z))$, which is not trivially true. And if $A$ is false, $z$ will obtain the knowledge from $B$ since the values of $x$ would be different after the analysis of branches $x := y$ and $x := z$.

B. The HM(TOP + SIMP) monitor

Le Guernic et al. [21] proposed a hybrid information flow monitor that uses static analysis for non-executed branches. The idea of the analysis is to compute a set of variables $modified$ that might be assigned to some value in the non-executed branch. Then, all the variables in $modified$ are tagged with a "high" label in case the test of the if-statement contained some 'high' variables. In a later work, Russo and Sabelfeld [29] define a generic framework of hybrid monitors where such syntactic checks are proposed as well.

To compare our monitors with the monitor of Le Guernic et al. [21], we propose a static analysis that sets the abstract value of a variable to $\top$ as soon as it gets assigned. By doing so, $\rho'(x) = \top$ means that $x \in modified$. Concretely, we take the static analysis from the HM(VAL + SIMP) monitor and substitute the [AAsignTop] rule with the following rule:

$$\text{[AAsignTop]} \quad \rho' = \rho[x \mapsto \top]$$

With this static analysis in mind, the idea of syntactic checks is already considered in our generic hybrid monitor. Whenever $x$ is in $modified$, its value will be $\top$, and hence according to the state combination procedure in Figure 5, the knowledge of the test will be added to the knowledge of $x$.

Theorem 6. HM(VAL + SIMP) monitor is more precise than HM(TOP + SIMP) monitor.
Table I: Non-interferent programs that are “accepted” (✓) or “rejected” (✗) by hybrid monitors

<table>
<thead>
<tr>
<th>Program 1</th>
<th>Program 2</th>
<th>Program 3</th>
<th>Program 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x := 0;</td>
<td>if B then x := 1</td>
<td>x := 1;</td>
<td>x := 1;</td>
</tr>
<tr>
<td>if B then y := 0</td>
<td>else x := 1;</td>
<td>y := 0;</td>
<td>y := 1;</td>
</tr>
<tr>
<td>else y := 1;</td>
<td>if A then z := 1</td>
<td>if (y = 1) then skip</td>
<td>if (y = 1) then skip</td>
</tr>
<tr>
<td>if (y = 1) then y := 0</td>
<td>else x := z;</td>
<td>else x := z;</td>
<td>else x := z;</td>
</tr>
<tr>
<td>output x;</td>
<td>output x;</td>
<td>output x;</td>
<td>output x;</td>
</tr>
</tbody>
</table>

HM(Val+Comb) ✓ ✓ ✓ ✓
HM(Val+Simp) ✓ ✓ ✓ ✓
HM(Top+Comb) ✓ X ✓ X
HM(Top+Simp) ✓ X X X

Program 2 from Table I illustrates the improved precision of the HM(Val+Simp) monitor compared to the HM(Top+Simp) monitor.

C. The HM(Top+Comb) monitor

In a later work, Le Guernic [20] proposed a more generic framework of hybrid monitors that use static analysis. One of the novelties of this work is that the static analysis should ignore the branch that will not be executed (according to the current environment) if the test before the branch does not contain any “high” variables. The formalization of this principle in our approach is a static analysis that uses the [AAssignTop] rule for assignment and [AIfComb] rule for if-statements.

Theorem 7. HM(Top+Comb) monitor is more precise than HM(Top+Simp) monitor.

In his PhD thesis [14], Le Guernic has proven that if a monitor on which we base HM(Top+Simp) monitor concludes that a variable x does not contain secret information, then the monitor similar to HM(Top+Comb) also concludes that variable x does not contain any secret information. Our framework generalises this proof since our notion of precision is based on the amount of knowledge in a variable.

Theorem 8. HM(Val+Comb) monitor is more precise than HM(Top+Comb) monitor.

To illustrate this precision result, consider the program 2 from Table I. The static analysis used by the HM(Top+Comb) monitor marks all the assigned variables as ⊤ because of the syntactic nature of [AAssignTop] rule. Hence, $\rho^r(x) = \top$, and so x will contain some knowledge about B.

Notice that programs 2 and 3 from Table I illustrate that HM(Top+Comb) and HM(Val+Simp) monitors are incomparable in a sense of their relative precision.

VIII. Discussion and Related work

A. Hybrid information flow monitoring

Hybrid monitors for information flow control that combine static and dynamic techniques have recently become popular [21], [20], [29], [26]. One of the first techniques was proposed by Le Guernic et al. [21] where the static analysis only performs syntactic checks on non-executed branches. This approach fits into our framework as HM(Top+Simp) monitor and it is proven to be less precise than the other monitors we propose. Russo and Sabelfeld [29] introduced a generic framework of hybrid monitors, where non-executed branches are also analysed only syntactically. In the follow-up work Le Guernic [20] presented a more permissive static analysis, that ignores possible branches that depend only on public variables. Inspired by this approach, we introduced HM(Top+Comb) monitor that is proven to be less precise than HM(Val+Comb) monitor.

Devriese and Piessens [11] proposed a secure multi-execution (SME) technique which falls outside of the static, dynamic or hybrid classification. Austin and Flanagan [3] introduced a faceted evaluation as another approach to the idea of secure multi-execution. The basic principle is to multi-execute the program for every security level while filtering inputs and outputs and thus enforcing non-interference. The approach was shown to be efficient in practice [10], when the security lattice contains only two levels: secret and public. It is not straightforward to compare the precision of secure multi-execution within our framework, however there are some similarities in the approaches. In case of only two security levels, the non-executed branch under the secret conditional will run in the other execution of SME, while it will be statically analysed in our hybrid monitor. More generally, in our setting each secret variable (browser feature) has a different security level (different knowledge), and combination of variables yields a creation of a new security level. In this case the security lattice grows exponentially (with the growth of a boolean formula) and SME approach would not be an efficient solution.
B. Non-interference

Our hybrid technique is monitoring the information flow (to be more precise, knowledge flow) from the secret (browser feature) variables to the public outputs. One of the possible applications of our monitor is to enforce non-interference. We can think about an enforcement by suppression of outputs that contain some knowledge (the formula is not trivially true). We can enforce termination-insensitive non-interference because our static analysis is termination-insensitive. Comparing to the other works in the area, termination-insensitive non-interference have been often a satisfactory formal guarantee for information flow monitors [1], [2], [3].

Another important property to consider is correction soundness introduced by Le Guernic [20]. A monitor is correction sound if on two executions that agree on public inputs, the low outputs get the same security level in the end of the execution. If the monitor enforcing non-interference is not correction-sound, it introduces new information leaks due to different enforcement reaction on different secret inputs. Our hybrid monitor would not obey this property, because when the program is non-interferent, there might be some secret inputs, for which the output would contains some information according to our monitor. The question of correction soundness worth a deeper investigation for monitors that track the knowledge flow of the program.

C. Quantitative information flow analysis

There are several approaches to quantify the information learned of the public observer about the secret program inputs. Clark, Hunt, Malacaria [17] propose a static analysis for quantitative information flow that calculates an upper and lower bound on the Shannon entropy. Backes et al. [5] compute the number of equivalence classes and their sizes by statically analysing the program, and compute the leakage using entropy-based measurements. Köpf and Rybalchenko [17] continue this line of work by computing over- and under-approximation on the size of equivalence classes. The main difference with these works is that we analyse a leakage of a concrete program execution, while they use entropy-based metrics for information leakage.

Clarkson, Myers and Schneider [8] define the belief of the observer about secret inputs as a probability distribution, and propose an algorithm to refine this belief with respect to the program that produces observable outputs. Based on this belief tracking approach, Mardziel et al. [22] propose an enforcement mechanism for knowledge-based policies. The knowledge of the observer is a probability distribution of secret variables, and the static analysis of the program makes a decision to run or reject the program. In case there exists a value of some secret variable that may increase the knowledge of the observer above some predefined threshold, the program is rejected. This enforcement is orthogonal to our approach, since the goal of our work is to dynamically evaluate program leakage of a concrete program execution as precise as possible. However, with our symbolic representation of knowledge in the variables we can parametrize our hybrid monitors with different enforcement techniques, and threshold-based enforcement is one of the first directions for future work. Also, Mardziel et al. keep history of the knowledge gained by the observer. This knowledge is updated whenever an observer sees more program output. Our knowledge representation would be a good starting point for investigating how to develop such a history-based approach in our setting.

To the best of our knowledge, the only dynamic analysis for quantitative information flow was proposed by McCamant and Ernst [24]. It uses a channel capacity metrics for information leakage. The channel capacity defines the smallest probability distribution, and hence puts an upper bound on the amount of leaked information. This approach is not precise enough in our setting since the probability distributions are known a-priori.

IX. Conclusions

Precise evaluation of information leakage in fingerprinting applications cannot be achieved by static, entropy-based quantitative information flow analysis. To obtain more precise evaluation of information leakage we have proposed hybrid analysis techniques based on computing a symbolic representation of the knowledge about secrets. We have developed a generic framework for hybrid monitors and proved generic results on its soundness, precision and requirements for the static analysis that a hybrid monitor uses. We instantiated the generic monitor with a combined static constant and dependency analysis and proved that this monitor is more precise than the other hybrid monitors found in the literature. The entire theory has been modelled and verified [25] using the Coq proof assistant.

For further work, Section VIII already discussed extensions towards correction soundness, threshold-based enforcement and inclusion of histories in the monitor. In addition, our hybrid analyses are defined for a simple programming language with focus on the principles behind the mechanism. We would have to scale to such languages as JavaScript for real deployment. Hedin and Sabelfeld [15] have shown it possible to analyse JavaScript using purely dynamic information flow technique. Their system seems an ideal candidate to instrument with our monitor in order to track and quantify the information a tracker can deduce about possible configurations by observing the program outputs.
REFERENCES


An information theoretic definition was used in the past to establish a theory of quantitative information flow. This theory has been developed in the works by Clark, Hunt and Malacaria [7] on static analysis for quantitative information flow, Clarkson, Myers and Schneider [8], Köpf and Basin [16] and others. Smith [31] proposes a framework for quantitative information flow; we will use definitions and notations from that paper.

When introducing the notations, we consider programs \( S \) with a high input \( H \) and a low output \( L \). For simplicity, we assume that there are no low inputs. We assume that \( H \) has a finite space of possible values \( \mathcal{H} \), and a probability of \( H \) having value \( h \) is denoted as \( P(H = h) \).

As opposed to the other works on quantitative information flow (QIF) [8], [22], [31], we will only consider deterministic programs because our long-term goal is to extend our approach to web scripts, such as JavaScript.

The goal now is to quantify how much information is leaked from \( H \) to \( L \). One of the common definitions is the Shannon entropy based definition of quantitative information flow for programs with no low input.

**Definition 11** (Shannon entropy based QIF). Given a program \( S \) with high security input \( H \) and low security output \( L \), the Shannon entropy based quantitative information flow is defined as

\[
SE(S) = \mathcal{I}(H; L) = H(H) - H(H|L) = H(L) - H(L|H)
\]

**APPENDIX**

### A. Information Theory

The probability distribution of the browser and OS properties, as well as their joint distribution is denoted by \( P \) and is constant throughout this paper. The sample spaces of the variables will be denoted by a corresponding letter, e.g. the sample space for \( X \) is \( \mathcal{X} \), for \( Y \) is \( \mathcal{Y} \) etc.

One of the central notions in information theory is self-information. It is a number of bits needed to encode a knowledge about the possible value of a random variable. For example, if \( X \) is a variable with possible values \( \mathcal{X} \), then the self-information about the fact that \( X \) has a value \( x \) is defined by

\[
I(X = x) = -\log_2 P(X = x)
\]

The notion of Shannon entropy [30] is just a weighted average of possible self-informations for all possible values of \( X \).

**Definition 8** (Shannon entropy). Let \( X \) be a random variable. The Shannon entropy of \( X \) is defined as

\[
H(X) = \sum_{x \in \mathcal{X}} P(X = x) \cdot I(X = x)
\]

The Shannon entropy can be explained as "uncertainty about \( X \)" , or an expected number of bits to transmit \( X \). Given two random variables, \( X \) and \( Y \), the conditional entropy denotes an uncertainty of \( X \) after knowing \( Y \).

**Definition 9** (Conditional entropy). Let \( X \) and \( Y \) be two random variables. Then, the conditional entropy of \( X \) given \( Y \) is defined as

\[
H(X|Y) = \sum_{y \in \mathcal{Y}} P(Y = y) \cdot H(X|Y = y)
\]

where

\[
H(X|Y = y) = \sum_{x \in \mathcal{X}} P(X = x|Y = y) \cdot I(X = x|Y = y)
\]

\[
P(X = x|Y = y) = \frac{P(X = x,Y = y)}{P(Y = y)}
\]

The mutual information between two random variables \( X \) and \( Y \) informally means "amount of information shared between \( X \) and \( Y \)".

**Definition 10** (Mutual information). Let \( X \), \( Y \) and \( Z \) be random variables. The mutual information of \( X \) and \( Y \) is defined as

\[
\mathcal{I}(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)
\]
First Clark et al. [7], and later Smith [31] have proposed a simplified definition of QIF in case of deterministic programs.

**Definition 12.** Given a deterministic program $S$ with high security input $H$ and low security output $L$, the Shannon entropy based quantitative information flow is defined as

$SE(S) = I(H;L) = H(L)$  \hspace{1cm} (9)

Smith [31] also proposed an alternative definition for quantitative information flow: Min-entropy that represents the smallest self-information that can be learned.

**Definition 13 (Min-entropy).** Let $X$ be a random variable. Then, the Min-entropy of $X$ is defined as

$H_\infty(X) = -\log_2 V(X)$  \hspace{1cm} (10)

and conditional min entropy of $X$ given $Y$ is defined as

$H_\infty(X|Y) = -\log_2 V(X|Y)$  \hspace{1cm} (11)

where

$V(X) = \max_{x \in X} P(X = x)$  \hspace{1cm} (12)

$V(X|Y = y) = \max_{x \in X} P(X = x|Y = y)$  \hspace{1cm} (13)

$V(X|Y) = \sum_{y \in Y} P(Y = y)V(X|Y = y)$  \hspace{1cm} (14)

For deterministic programs, Smith proposed a min-entropy-based definition of QIF.

**Definition 14 (Min-entropy based QIF).** Given a deterministic program $S$ with high security input $H$ and low security output $L$, the Min-entropy based quantitative information flow is defined as

$ME(S) = H_\infty(H) - H_\infty(H|L)$  \hspace{1cm} (15)

As Smith later noticed [32], that it might seem tempting to denote Min-entropy based QIF as $I_\infty(H;L)$, by analogy with mutual information. However, it cannot be adopted because Min-entropy based QIF is not symmetric.

There are two other definitions of entropy used in the theory of QIF. First, Guessing entropy of $X$ given $Y$ represents an average number of times required for the attacker to guess $X$ given that he knows $Y$. We will not consider this definition since it is another averaged value that is not precise enough for a particular browser setting. Second, Channel capacity is simply a maximum Shannon-entropy based QIF over all possible distributions. In our setting the probability distribution is known in advance, and hence Channel capacity is not applicable.

$^5$This equation holds because the low output $L$ is predefined by the high input $H$ in case of deterministic program, i.e. $H(L|H) = 0.$