Homework 3

Carlo Leo 546155

June 2021

1 Answers

In this section is reported the answers to questions about CFA.

1.1 CFA and its application to Pi Calculus

1. In order to put up an over-approximation, extracting abstract behaviour from program text, the analysis needs to know two sets of information:

   - possible communications of a process
   - what can be communicated by a process

   Information is provided through two functions: ρ and κ, where:

   - ρ is a function such that given a name returns the set of names which can be bound to it, and ρ(n) = n for every free name n.
   - κ is a function such that given a name, returns the set of names which can be sent over it.

2. Considering: $P \mid Q \mid R \;=\; \overline{a}(d).P' + \overline{a}b.P''\mid R \mid a(w).\overline{w}c.Q'$ then analysis will compute: ρ(a) = {a}, ρ(b) = {b}, ρ(d) = {d}, ρ(c) = {c}, ρ(w) = {d, b} and κ(a) = {b, d}, κ(b) = {c}, κ(d) = {c}, κ(c) = $\emptyset$.

Now assuming that names are gathered as follow:

- *SecretNames* = {a, b}
- *PublicNames* = {b, c}

Via the control flow analysis we will realise that the program will not disobey the secrecy property when it runs, since there is no secret data which flows over public channel into estimate computed. CFA is able to claim that, because it computes an over-approximation of all possible process behaviours.
Formally, the static notion of confinement is obeyed: $\kappa($ n $\in$ P$) \subseteq$ *PublicNames*. Relying on the following theorem:

Theorem 1 *If a system S is confined then S is careful*

we can conclude that it satisfies the dynamic notation of carefulness.

1.2 CFA and its application to Lysa and IoT-Lysa

1. The error component ψ consists in a set of couples like $(\ell, \ell\prime)$, which represents all possible mismatches of origin and destination labels found out by CFA. For instance the couple $(\ell, \ell\prime)$ indicates that something encrypted at ℓ was unexpectedly decrypted at $\ell\prime$.
At the end of the analysis if ψ is not equal to $\emptyset$ then the analysis warns about possible attack sequences. So to be on the safe side, the process P can be executed using reference monitor semantics which will abort the execution as soon as annotations are violated. (e.g it is expecting sender equal to A but comes equal to something else)

2. The main differences between the CFA estimates of LySa and those of IoT Lysa are being listed below.
First of all, to keep trace of all possible values which can be bound to terms, CFA, in case of Iot LySa, does not compute ρ as it does in case of LySa. That because, IoT LySa is used in a hierarchical context where communication is being done between nodes and each node owns a locally memory (node components communicate between them via

shared local memory). To take into account that, in case of IoT LySa CFA, estimates contain two new component:

- $\widehat{\Sigma}$: which provides an abstraction of data a node may store
- $\widehat{\Theta}$: which provides an abstraction of data a node may compute

Estimates of LySa, unlike IoT LySa, also contain the error component ψ, which is explained at point 1. Both types of estimate keep containing the network component κ. Last one consists in the use of abstract values by CFA in case of Iot LySa (to avoid considering all concrete ones) unlike in the case of LySa wherein all possible values for a given variable are considered.

3. Abstract values, being bound to a node label (each node is identified by a unique label), allow us to understand how the information is being managed within the network. Precisely, carrying origin node label up to supervisor nodes, they make possible to track information flow. In this way regardless of real value, it is possible through CFA result to verify security properties of values at critical point of the system where they are used into actuation decisions. An example is CFA taint analysis, which allows to check trustworthy of values.

2 Framework extension

In this section is explained a proposed solution for the exercise *Taint and Resilience Analysis a là Quality*.

Our approach leverages on adding a label ι to abstract values, which turn into:

$(\top, b, \iota)$

(υ, b, ι)

$(f(\widehat{\upsilon_1}, \cdots, \widehat{\upsilon_r}), b, \iota)$

$(\{\widehat{\upsilon_1}, \cdots, \widehat{\upsilon_r}\}_k, b, \iota)$

$where\ \iota \in \{Hi,\ Lo\}\ and\ Lo\ <\ Hi$

The effective label will be assigned according to the node which values come from.

So we assume that nodes are divided in two sets:

- N_H: high level nodes
- N_L: low level nodes

As a consequence, to take into account all new information we have to modify some definition.

The *data classification* function needs to be changed.

Definition 1 *(Data classification) Given the set of sensitive sensors S_ℓ, the set of the tamperable sensors T_ℓ, the set of high level nodes N_H and the set of low level nodes N_L, the taint assignment function τ is defined as follow:*

$$\tau(y,\ell) = \begin{cases} (\text{⟠}, Hi) & \textit{if } y \in S_\ell \wedge \ell \in N_H \\ (\text{⟠}, Lo) & \textit{if } y \in S_\ell \wedge \ell \in N_L \\ (\text{⟡}, Lo) & \textit{if } y \in T_\ell \ \wedge \ell \in N_L \\ (\text{⟡}, Hi) & \textit{if } y \in T_\ell \ \wedge \ell \in N_H \\ (\text{◈}, Lo) & \textit{if } y \in T_\ell \cap S_\ell \wedge \ell \in N_L \\ (\text{◈}, Hi) & \textit{if } y \in T_\ell \cap S_\ell \wedge \ell \in N_H \\ (\diamond, Hi) & \textit{if } y \notin T_\ell \cup S_\ell \ \ell \in N_H \\ (\diamond, Lo) & \textit{if } y \notin T_\ell \cup S_\ell \ \ell \in N_L \end{cases}$$

To correctly propagate taint information together with quality information, also the combination operator changes as follow:

Definition 2 *(combination operator [1])*
$\otimes' : (B \times I) \times (B \times I) \to (B \times I), and \ \otimes'(b, \iota, b', \iota') = (\otimes(b, b'), \otimes(\iota, \iota'))$
where $\otimes$ keeps working as join operator.

Once the quality has been introduced within the framework, we can also extend syntax of IoT Lysa embedding quality logic predicates. To do so, terms add a member, as shown below:
$E \ := v \mid i \mid x \mid f(E_1, \ldots, E_n) \mid f(\&_{m/n}(E_1, \ldots, E_n)) \ with \ 1 <= \ m \ <= n$
The quality logical predicate, $\&_{m/n}$, obeys its normal semantic, namely it checks that at least $\frac{m}{n}$ input parameters of the function f have high quality. High quality concept is defined in definition 3.

[1] As a consequence of definition 2, all CFA clauses replace $\otimes$ with $\otimes'$

Definition 3 *(High quality) An abstract value $\widehat{\upsilon_i}$ has high quality if and only if $\widehat{\upsilon_i}\downarrow_3 = Hi \vee \widehat{\upsilon_i}\downarrow_2 \in \{\diamondminus,\ \Diamond\}$.*

When the quality logical predicate is being evaluated to false then the static analyzer binds to abstract value, which represents the function, the couple ($\diamondvert$, Lo) as b and ι respectively.

Whereas, when the quality logical predicate is being evaluated to true, then the analyzer bind to abstract value, which represents the function, the couple ($\Diamond$, q) as b and ι respectively, where q is equal to the most frequent quality tag among arguments of the function.
In this way we introduce the fault tolerance inside our framework since a function will not be tagged with the maximum taint tag of its arguments, but in case the quality logical predicate returns true , the function will be assigned minimum taint tag: *untainted* (arguments outside the scope of the quality logical predicate can have a bigger taint tag) . Formally it is explained by definition 4. On the other hand, by parameter m it is possible to set the granularity in order to be conformed to critical points.
The *taint propagation policy* has to change since, extending abstract values with quality label ι, the result of the application of both function f and encryption function, besides taint information, carries the quality information as well. Furthermore, we have to define how information about the taint and quality propagate for the quality logical predicate.

Definition 4 *(taint propagation policy)*
Given the combination operator $\otimes\prime$, the taint and quality resulting by the application of

- *a function f is $F_\tau(f, \widehat{\upsilon_1}, \cdots, \widehat{\upsilon_r}) = \otimes\prime((\widehat{\upsilon_1}\downarrow_2, \widehat{\upsilon_1}\downarrow_3), \cdots, (\widehat{\upsilon_r}\downarrow_2, \widehat{\upsilon_r}\downarrow_3))$*

- *a function $f_\&$ is $F_\tau(f, \&, \widehat{\upsilon_1}, \cdots, \widehat{\upsilon_r}) =$*

$$\begin{cases} (\Diamond, q) \text{ if } \& \text{ holds} \\ (\diamondvert, Lo) \quad \text{otherwise} \end{cases}$$

  with $q = max_{Hi,Lo}(\widehat{\upsilon_1}, \cdots, \widehat{\upsilon_r})$

- *an encryption function is*

$$Enc_\tau(\widehat{v}_1, \cdots, \widehat{v}_r) = \begin{cases} (◇, Hi) & \textit{if } \forall i\ \widehat{v}_i \downarrow_2 \in \{⟠, ◇\} \wedge \forall i \widehat{v}_i \downarrow_3 = Hi \\ (◇, Lo) & \textit{if } \forall i\ \widehat{v}_i \downarrow_2 \in \{⟠, ◇\} \wedge \exists i\ \widehat{v}_i \downarrow_3 = Lo \\ (⧫, Hi) & \textit{if } \exists i\ \widehat{v}_i \downarrow_2 \notin \{⟠, ◇\} \wedge \widehat{v}_i \downarrow_3 = Hi \\ (⧫, Lo) & \textit{if } \exists i\ \widehat{v}_i \downarrow_2 \notin \{⟠, ◇\} \wedge \widehat{v}_i \downarrow_3 = Lo \end{cases}$$

Combining the definitions (1, 2, 4), as a result, we get that the framework is able to to track quality information as well as taint information, that basically can be exploited to have some grade of tolerance during **static taint analysis**.
With this idea, we reshape *definition 2.7* and *theorem 2.8* shown during last lecture in order to take into account quality information.

Definition 5 *Let N be a system of nodes with labels in L, $S = \{S_\ell \mid \ell \in L\}$ the set of its sensitive sensors , $T = \{T_\ell \mid \ell \in L\}$ the set of its tamperable sources, and P a set of program critical points. Then N does not use tainted values in a critical point if $N \rightarrow_* N\prime$ and there is no transition $N\prime \xrightarrow{M_1^1, \cdots, M_r^r} N\prime\prime$ s.t.*
$a_i \in P$ *and* $([[M_i^{a_i}]]_{\Sigma_\ell^i} \downarrow_2 \subseteq \{ ⟠, ⧫, ◈\} \wedge [[M_i^{a_i}]]_{\Sigma_\ell^i} \downarrow_3 \subseteq \{Lo\})$ *for some* $i \in \{1, \cdots, r\}$ *and* ℓ *in* L

Theorem 2 *Let N be a system of nodes with labels in L, $S = \{S_\ell \mid \ell \in L\}$ the set of its sensitive sensors , $T = \{T_\ell \mid \ell \in L\}$ the set of its tamperable sources, and P a set of program critical points, then N does not use tainted values in a critical point if*
$(\widehat{\Sigma}, \kappa, \Theta) \models N$ *and*
$[\ (\Theta(\ell)(a) \downarrow_2 = \{◇\}\ \vee (\Theta(\ell)(a) \downarrow_2 \in \{⟠, ⧫, ◈\} \wedge \Theta(\ell)(a) \downarrow_3 = Hi\ )\ ]$
for all labels $a \in P$ *and* $\ell \in L$

For the way things are, when the verification of a quality predicate fails, it will lead to a situation where a wrong value is used at some critical point (it will be part of $\Theta(\ell)(a)$). As a result, the conditions of the theorem 2 will not fulfilled, so at static time the framework will state that the system could use unsafe data at some critical point, when it runs.
The quality component make the analysis less strict so the overall system becomes more flexible. We have done that, assuming high level nodes act like **trust entity**.
To conclude, via this extension the framework provides a bigger flexibility supporting resilience but, the properties guaranteed are always secure.