set of private constraints of all those on the variable parameter or intuitively:

$$\text{reset}(C, x) = C \mid \{(x, \cdot, \cdot) (\cdot, \cdot, x)\}$$

By abusing notation, we extend this function to the sets of variables:

$$\text{reset}(C, X) = C \mid (\{(x, \cdot, \cdot) \mid x \in X\} \cup \{(\cdot, \cdot, x) \mid x \in X\})$$

V_{assign}

- $v \in V_{assign}$ of shape $x = \alpha$ with $x \in Vars_Z$ $\alpha \in Vars_Z \cup Z$

$$(C, P)_{v, c(} \text{reset}(C, x) \cup \{(x =, \alpha)\} \cup \{c\}, P)$$

In an a? Single assignment of an integer variable to another variable or an integer constant is removed from the designed abstract configuration all constraints on x and a new equality constraint is added. Function P giving for each destination pointer is not modi? ed. It also adds the stress carried by the arc that follows the node, as in all rules.

- $v \in V_{assign}$ of shape $p = q$ with $p \in Vars_{ptr}$, $q \in Vars_{ptr}$

$$(C, P)_{v, c(} \text{reset}(C\, p) \cup \{(p = q)\} \cup \{c\},$$
$$P\,[p \leftarrow P(q)])$$

This rule has a case similar to the previous but this time, a pointer sees a? Ected the value of another pointer. We apply the same modi? Cations of the designed configuration as in the previous rule and change? E and more function P to indicate that p has now the same destination q.

- $v \in V_{assign}$ of shape $p = \&\, v$ with $p \in Vars_{ptr}$, $v \in Vars_{same}$

$$(C, P)_{v, c(} \text{reset}(C\, p) \cup \{c\}, P\,[p \leftarrow v])$$

In the case where a pointer p sees a? ected the address of a variable v constraints on we suppress p but it changes the function P to note the new association between p and v.

- $v \in V_{assign}$ of shape $x = {}^*q$ with $x \in Vars_Z$ $q \in Vars_{ptr}$ and $P(q) = z \in Vars_{Z(}$ and so $P(q)\ 6 =>)$

$$(C, P)_{v, c(} \text{reset}(C, x) \cup \{(x =, z)\} \cup \{c\}, P)$$

If is assigned to an integer variable dereference a pointer q and if the destination $P(q)$ This pointer is known and is a whole then we can add the equality constraint between the variable and the variable pointed.

- $v \in V_{assign}$ of shape $x = *q$ with $x \in Vars_Z$ $q \in Vars_{ptr}$ and $P(q) = z \in Vars_{ptr}$ or $P(q) = >$

$$(C, P)_{v,c(}\text{reset}(C, x) \cup \{c\}, P)$$

However, if the destination $P(q)$ pointer is a pointer (integer to pointer cast) or is unknown then only removes constraints on x because its new value is unknown.

- $v \in V_{assign}$ of shape $p = *q$, $p \in Vars_{ptr}$, $q \in Vars_{ptr}$ with $P(q) = r \in Vars_{ptr}$

$$(C, P)_{v,c(}\text{reset}(C\,p) \cup \{(p, =, r)\} \cup \{c\}, P[p \leftarrow r])$$

When a pointer is a? Ected dereference a pointer to a pointer, the pointed value is known P we update the set of constraints with the new equality and P.

- $v \in V_{assign}$ of shape $p = *q$ with $p \in Vars_{ptr}$, $q \in Vars_{ptr}$ and $P(q) = >$

$$(C, P)_{v,c(}\text{reset}(C\,p) \cup \{c\}, P[p \leftarrow >])$$

- $v \in V_{assign}$ of shape $x = e$ with $x \in Vars_Z$ and $e = @? \cup Vars_{ptr}$

$$(C, P)_{v,c(}\text{reset}(C, x) \cup \{c\}, P)$$

When a? Assignment of an integer variable is made with an unknown right side value or a pointer (in the case of casting), the constraints on the left side variable are deleted.

- $v \in V_{assign}$ of shape $p = e$ with $p \in Vars_{ptr}$ and $e \in @? \cup Vars_Z \cup Z$

$$(C, P)_{v,c(}\text{reset}(C\,p) \cup \{c\}, P[p \leftarrow >])$$

When a? Assignment of a pointer is made with an unknown value in right or an integer value (in the case of casting), the constraints on the left side variable are deleted.

- $v \in V_{assign}$ of shape $w = e$ with $w \in Vars$

$$(C, P)_{v,c(}C \cup \{c\}, P)$$

If a? Assignment has left a partially identi? Ant that is not part of the variables taken into account, the con? Guration is not modi? Ed by the node.

V_{same}

- $v \in V_{same}$ of shape $*p = \alpha$ with $p \in Vars_{ptr}$, $\alpha \in Vars_Z \cup Z$ and $P(p) = x \in Vars_Z \cap Vars_{same}$

$$(C, P)_{v,c(}\text{reset}(C, x) \cup \{(x =, \alpha)\} \cup \{c\}, P)$$

One has? Assignment through a pointer p which points to a variable x fully known to add an equality constraint between the variable pointed and right. Function P is not altered.

- $v \in V_{same}$ of shape $*p = e$ with $p \in Vars_{ptr}$, $e \in Vars \cup \mathbb{Z}$ $P(p) = >$ and $Oracle(p) \subseteq Vars_{\mathbb{Z}}$

$$(C, P)_{v,c} (reset(C Oracle(p)) \cup \{c\}, P)$$

If the variable pointed to by p is unknown, delete the designed configuration all constraints on a possibly variable pointed by p, after the alias GCC oracle. This case is when p point to integers.

- $v \in V_{same}$ of shape $*p = r$ with $p \in Vars_{ptr}$, $r \in Vars_{ptr}$ and $P(p) = q \in Vars_{ptr} \cap Vars_{same}$

$$(C, P)_{v,c} (\text{reset}(C, q) \cup \{(q, =, r)\} \cup \{c\}, P[q \leftarrow P(r)])$$

If the pointer p is a pointer to a pointer, the equality constraint is added and modified? e function P Consequently.

- $v \in V_{same}$ of shape $*p = e$ with $p \in Vars_{ptr}$ and $P(p) = >$

$$(C, P)_{v,c} (reset(C Oracle(p)) \cup \{c\}, P[q \leftrightarrow > | q \in Oracle(p) \cap Vars_{ptr})$$

If the variable pointed to by p is unknown and p is a pointer to a pointer, it also implies modi? er function P in addition to removing constraints C.

- $v \in V_{same}$ of shape $*p = e$ with $p \in Vars_{ptr}$, $P(p) = x \in Vars_{\mathbb{Z}}$ and $e \in \mathbb{Z}? \cup Vars_{ptr}$

$$(C, P)_{v,c} (\text{reset}(C, x) \cup \{c\}, P)$$

Yes p points to the variable x and e is an unknown value or to a pointer (in case of casting), the constraints on is removed x.

- $v \in V_{same}$ of shape $*p = e$ with $p \in Vars_{ptr}$, $P(p) = > Oracle(p) \subseteq Vars_{\mathbb{Z}}$ and $e \in \mathbb{Z}?$

$$(C, P)_{v,c} (\text{reset}(C Oracle(p)) \cup \{c\}, P)$$

Yes p points to an unknown integer variable and e is a naked incon- value, the variables which all constraints are removed to be pointed by p, after the alias GCC oracle.

- $v \in V_{same}$ of shape $*p = e$ with $p \in Vars_{ptr}$, $P(p) = q \in Vars_{ptr}$ and $e \in \mathbb{Z}? \cup Vars_{\mathbb{Z}} \cup \mathbb{Z}$

$$(C, P)_{v,c} (\text{reset}(C, q) \cup \{c\}, P[q \leftrightarrow >])$$

Yes p points to a pointer q and e is an unknown value or an integer value (in case of casting), the constraints on is removed q and modi? e accordingly function P.

- $v \in V_{same}$ of shape $*p = e$ with $p \in V ars_{ptr}$, $P(p) =>$ and $e \in @? \cup V ars_Z \cup Z$

$$(C, P)_{v,c(} \text{reset} (C\ Oracle(p)) \cup \{c\},$$
$$P[q \leftrightarrow > | q \in Oracle(p)])$$

Yes p points to a pointer unknown and that e is an unknown value, the stress is removed on all the pointers which p could point and change? e accordingly function P.

V_φ In all these cases, $e_{path} \in \{e_1 \ldots, e_{not}\}$ is an expression among the arguments of the node φ corresponding to the path analyzed current. Knowing what argument is used in each path is provided by CCG is an intrinsic property of the path.

- $v \in V_\varphi$ of shape $x =$ IHP $< e_1 \ldots, e_{not} >$ or $x \in V ars_Z$ and $e_{path} \in Z \cup V ars_Z$

$$(C, P)_{v,c(} \text{reset} (C, x) \cup \{(x = e_{path})\} \cup \{c\}, P)$$

A knot φ essentially acts as a node has? assignment.

- $v \in V_\varphi$ of shape $x =$ IHP $< e_1 \ldots, e_{not} >$ or $x \in V ars_Z$ and $e_{path} \in @? \cup V ars_{ptr}$

$$(C, P)_{v,c(} \text{reset} (C, x) \cup \{c\}, P)$$

- $v \in V_\varphi$ of shape $p =$ IHP $< e_1 \ldots, e_{not} >$ or $p \in V ars_{ptr}$ and $e_{path} \in V ars_{ptr}$

$$(C, P)_{v,c(} \text{reset} (C, x) \cup \{< x = e_{path})\} \cup \{c\},$$
$$P[x \leftarrow P(e_{path})])$$

- $v \in V_\varphi$ of shape $p =$ IHP $< e_1 \ldots, e_{not} >$ or $p \in V ars_{ptr}$ and $e_{path} = \&\ there$ with $there \in V ars_{same}$

$$(C, P)_{v,c(} \text{reset} (C, x) \cup \{c\}, P[x \leftarrow y])$$

- $v \in V_\varphi$ of shape $p =$ IHP $< e_1 \ldots, e_{not} >$ or $p \in V ars_{ptr}$ and $e_{path} = @? \cup V ars_Z \cup Z$

$$(C, P)_{v,c(} \text{reset} (C, x) \cup \{c\}, P[x \leftrightarrow >])$$

V_{call}

- $v \in V_{call}$ of shape $x = f(e_1 \ldots, e_{not})$ with $x \in Vars_Z$

$$(C, P)_{v, c(} \text{reset} (C, \{x\} \cup Vars_{same)} \cup \{c\},$$
$$P[q \leftrightarrow / q \in Vars_{same} \cap Vars_{ptr})$$

A function can contain any kind of instructions. Therefore, it is impossible to predict the impact on the values of variables that the function call leads. We e? Ace therefore all constraints on memory alive variables, including pointers. Variable $Vars_{temp}$ are not impacted because they are specific to each function.

- $v \in V_{call}$ of shape $p = f(e_1 \ldots, e_{not})$ with $p \in Vars_{ptr}$

$$(C, P)_{v, c(} \text{reset} (C \{p\} \cup Vars_{same)} \cup \{c\},$$
$$P[q \leftrightarrow / q \in (Vars_{same} \cap Vars_{ptr)} \cup p])$$

- $v \in V_{call}$ of shape $w = f(e_1 \ldots, e_{not})$ with $w /$ $\in Vars$

$$(C, P)_{v, c(} \text{reset} (C \{w\} \cup Vars_{same)} \cup \{c\},$$
$$P[q \leftrightarrow / q \in Vars_{same} \cap Vars_{ptr})$$

- $v \in V_{call}$ of shape $f(e_1 \ldots, e_{not})$

$$(C, P)_{v, c(} \text{reset} (C \{w\} \cup Vars_{same)} \cup \{c\},$$
$$P[q \leftrightarrow / q \in Vars_{same} \cap Vars_{ptr})$$

V_{join}

- $v \in V_{join}$

$$(C, P)_{v, c(} C \cup \{c\}, P)$$

The junction nodes modi? ent not designed current configuration. In Coq, unlike the concrete semantics, we can de? Ne com- pletely the relationship.

```
1 inductive AbstractSemantics: AbstractConfiguration → nodes → constraint
        →  AbstractConfiguration → Prop : =
  | asem_assignZ_to_Varz CP (x: Vars) (Hnptr_x: ¬point x) (z: Z) c:
     AbstractSemantics (C, P) (X assignZ_to_Varz Hnptr_x z) c
      ( set_add Constraint_eq_dec (constraint1 Eq x z)
5       ( set_add Constraint_eq_dec c (reset C x)), P)
  | asem_assignVarz_to_Varz CP (x: Vars) (Hnptr_x: ¬point x)
     ( v: Vars) (Hnptr_v: ¬point v) c: AbstractSemantics (C, P) (X assignVarz_to_Varz Hnptr_x Hnptr_v v) c

      ( set_add Constraint_eq_dec (constraint2 x Eq v)
10      ( set_add Constraint_eq_dec c (reset C x)), P)
  | asem_assignPtr_to_Ptr CP (x: Vars) (Hptr_x: point x)
     ( e: Vars) (Hptr_e: point e) c Hconsistency: AbstractSemantics (C, P) (X assignPtr_to_Ptr Hptr_x e Hptr_e
     Hconsistency) c
```

```
      ( set_add Constraint_eq_dec (constraint2 Eq x e)
        ( set_add Constraint_eq_dec c (reset C x)),
        ( update_with_existing_element P x Hptr_x Hptr_e Hconsistency e))
| asem_assignAddr_to_Ptr CP (x: Vars) (Hptr_x: point x)
    ( y: Vars) (Haddr_y: addressable y) c Hconsistency: AbstractSemantics (C, P) (X assignAddr_to_Ptr Hptr_x there
    Haddr_y Hconsistency) c
      ( set_add Constraint_eq_dec c (reset x C),
        ( update_with_address P x y Hptr_x Haddr_y Hconsistency))
| asem_assignDerefPtr_to_Varz CP
    ( x: Vars) (Hnptr_x: ¬ point x) (p: Vars) (Hptr_p: pointer p) c
    ( v: Vars) (Haddr_v: addressable v)
    ( hp: (Proj1_sig P) = p Hptr_p ptvar Haddr_v v) pHPT: AbstractSemantics (C, P) (X assignDeref_to_Varz
    Hnptr_x p Hptr_p pHPT) c
      ( set_add Constraint_eq_dec (constraint2 x Eq v)
        ( set_add Constraint_eq_dec c (reset C x)), P)
| asem_assignDerefUnknown_to_Varz CP
    ( x: Vars) (Hnptr_x: ¬ point x)
    ( p: Vars) (Hptr_p: point p) c
    ( hp: (Proj1_sig P) = p Hptr_p ptunknown) Horacle: AbstractSemantics (C, P) (X assignDeref_to_Varz Hnptr_x p
    Hptr_p Horacle) c
      ( set_add Constraint_eq_dec c (reset x C), P)
| asem_assignDeref_to_Ptr CP (x: Vars) (Hptr_x: point x)
    ( p: Vars) (Hptr_p: pointer p) c (v: Vars) (Haddr_v: addressable v)
    ( hp: (Proj1_sig P) = p Hptr_p ptvar Haddr_v v) pHPT Hconsistency: AbstractSemantics (C, P) (X
    assignDeref_to_Ptr Hptr_x p Hptr_p pHPT

        Hconsistency) c
      ( set_add Constraint_eq_dec (constraint2 x Eq v)
        ( set_add Constraint_eq_dec c (reset C x)),
          ( update_with_existing_element P x v Hptr_x
            ( PHPT _ _ ((proj2_sig P) _ _ _ _ Hp))
            ( Hconsistency _ _ _ ((proj2_sig P) _ _ _ _ Hp))))
| asem_assignOther_to_Varz CP (x: Vars) (Hnptr_x: ¬ point x) (e: OtherExprs) c:
    AbstractSemantics (C, P) (X assignOther_to_Varz Hnptr_x e) c
      ( set_add Constraint_eq_dec c (reset x C), P)
| asem_assignPtr_to_Varz CP (x: Vars) (Hnptr_x: ¬ point x)
    ( e: Vars) (Hptr_e: point e) c:
    AbstractSemantics (C, P) (X assignPtr_to_Varz Hnptr_x Hptr_e e) c
      ( set_add Constraint_eq_dec c (reset x C), P)
| asem_assignDerefUnknown_to_Ptr CP (x: Vars) (Hptr_x: point x)
    ( p: Vars) (Hptr_p: pointer p) c (Hp (proj1_sig P) = p Hptr_p ptunknown) pHPT Hconsistency:

    AbstractSemantics (C, P)
      ( assignDeref_to_Ptr x Hptr_x p Hptr_p pHPT Hconsistency) c
        ( set_add Constraint_eq_dec c (reset x C),
          ( update_with_top Hptr_x P x))
| asem_assignOther_to_Ptr CP (x: Vars) (Hptr_x: point x) (e: OtherExprs) c:
    AbstractSemantics (C, P) (X assignOther_to_Ptr Hptr_x e) c
      ( set_add Constraint_eq_dec c (reset x C),
        ( update_with_top Hptr_x P x))
| asem_assignZ_to_Ptr CP (x: Vars) (Hptr_x: point x) (E: Z) c:
    AbstractSemantics (C, P) (X assignZ_to_Ptr Hptr_x e) c
      ( set_add Constraint_eq_dec c (reset x C),
        ( update_with_top Hptr_x P x))
```

```
| asem_assignVarz_to_Ptr CP (x: Vars) (Hptr_x: point x)
    ( e: Vars) (Hnptr_e: ¬ point e) c: AbstractSemantics (C, P) (X assignVarz_to_Ptr Hptr_x Hnptr_e e) c

      ( set_add Constraint_eq_dec c (reset x C),
        ( update_with_top Hptr_x P x))
| asem_assign_to_IgnoredVar CP (x: IgnoredVars) (e: Exprs) c:
    AbstractSemantics (C, P) (Assign_to_IgnoredVar xe) c (set_add
        Constraint_eq_dec c C, P)

| asem_memZ_to_Ptr CP (p Vars) (Hptr_p: pointer p) (z: Z) c
    ( v: Vars) (Haddr_v: addressable v)
    ( PHPT: (Proj1_sig P) = p Hptr_p ptvar Haddr_v v) Hnonpointers: AbstractSemantics (C, P) (P
    memZ_to_Ptr Hptr_p Hnonpointers z) c
       ( set_add Constraint_eq_dec (constraint1 v Eq z)
         ( set_add Constraint_eq_dec c (reset C v)), P)
| asem_memVarz_to_Ptr CP (p Vars) (Hptr_p: point p)
    ( e: Vars) (Hnptr_e: ¬ point e) c (v Vars) (Haddr_v: addressable v)
    ( PHPT: (Proj1_sig P) = p Hptr_p ptvar Haddr_v v) Hnonpointers: AbstractSemantics (C, P) (P memVarz_to_Ptr
    Hptr_p e Hnptr_e Hnonpointers) c
       ( set_add Constraint_eq_dec (constraint2 v Eq e)
         ( set_add Constraint_eq_dec c (reset C v)), P)
| asem_memZ_to_Unknown CP (p Vars) (Hptr_p: pointer p) (z: Z) c
    ( PHPT: (Proj1_sig P) = p Hptr_p ptunknown) Hnonpointers: AbstractSemantics (C, P) (P
    memZ_to_Ptr Hptr_p Hnonpointers z) c
       ( set_add Constraint_eq_dec c (C p reset_all_pt Hptr_p) P)
| asem_memVarz_to_Unknown CP (p Vars) (Hptr_p: point p)
    ( e: Vars) (Hnptr_e: ¬ point e) c
    ( PHPT: (Proj1_sig P) = p Hptr_p ptunknown) Hnonpointers: AbstractSemantics (C, P) (P memVarz_to_Ptr Hptr_p
    e Hnptr_e Hnonpointers) c
      ( set_add Constraint_eq_dec c (C p reset_all_pt Hptr_p) P)
| asem_memPtr_to_Ptr CP
    ( p: Vars) (Hptr_p: point d) (e Vars) (Hptr_e: point e) c
    ( v: Vars) (Haddr_v: addressable v)
    ( PHPT: (Proj1_sig P) = p Hptr_p ptvar Haddr_v v)
    ( Hpointers: pointers_to_pointers p Hptr_p) Hconsistency: AbstractSemantics (C, P)

    ( memPtr_to_Ptr p Hptr_p e Hptr_e Hpointers Hconsistency) c
      ( set_add Constraint_eq_dec (constraint2 v Eq e)
        ( set_add Constraint_eq_dec c (reset C v)),
          ( update_with_existing_element P v (Hpointers _ _ ((proj2_sig P)
           _ _ _ _ pHPT)) e Hptr_e

            ( Hconsistency _ _ _ ((proj2_sig P) _ _ _ _ pHPT))))
| asem_memPtr_to_Unknown CP (p Vars) (Hptr_p: point p)
    ( e: Vars) (Hptr_e: point e) c
    ( PHPT: (Proj1_sig P) = p Hptr_p ptunknown)
    ( Hpointers: pointers_to_pointers p Hptr_p) Hconsistency: AbstractSemantics (C, P)

    ( memPtr_to_Ptr p Hptr_p e Hptr_e Hpointers Hconsistency) c
       ( set_add Constraint_eq_dec c (C p reset_all_pt Hptr_p)
         ( update_all_pt_with_top Hptr_p P p))

| asem_memUnknownVarz_to_Ptr CP (p Vars) (Hptr_p: point p)
   ( e: OtherExprs) c
```

```
120    ( v: Vars) (Haddr_v: addressable v) (Hnptr_v: ¬point v)
       ( PHPT: (Proj1_sig P) = p Hptr_p ptvar Haddr_v v): AbstractSemantics
         (C, P)
          ( memOther_to_Ptr p Hptr_p e) c
            ( set_add Constraint_eq_dec c (reset C v), P)
125  | asem_memUnknownVarz_to_Unknown CP (p Vars) (Hptr_p: point p)
       ( e: OtherExprs) c
       ( PHPT: (Proj1_sig P) = p Hptr_p ptunknown)
       ( Hpointers: pointers_to_non_pointers p Hptr_p): AbstractSemantics (C, P)

130      ( memOther_to_Ptr p Hptr_p e) c
           ( set_add Constraint_eq_dec c (C p reset_all_pt Hptr_p) P)

     | asem_memUnknownPtr_to_Ptr CP (p Vars) (Hptr_p: point p)
       ( e: OtherExprs) c
135    ( q: Vars) (Haddr_q: addressable q) (Hptr_q: point q)
       ( PHPT: (Proj1_sig P) = p Hptr_p ptvar q Haddr_q): AbstractSemantics
         (C, P)
          ( memOther_to_Ptr p Hptr_p e) c
            ( set_add Constraint_eq_dec c (reset C q),
140          update_with_top Hptr_q P q)
     | asem_memUnknownPtr_to_Unknown CP (p Vars) (Hptr_p: point p)
       ( e: OtherExprs) c
       ( PHPT: (Proj1_sig P) = p Hptr_p ptunknown):
         AbstractSemantics (C, P)
145      ( memOther_to_Ptr p Hptr_p e) c
           ( set_add Constraint_eq_dec c (C p reset_all_pt Hptr_p) update_all_pt_with_top
             Hptr_p P p)

     | asem_mem_to_IgnoredVars CP (w: IgnoredVars) (e: Exprs) c:
150    AbstractSemantics (C, P)
         ( mem_to_IgnoredVar we) c (c set_add Constraint_eq_dec C, P)

     | asem_phiZ_to_Varz CP (x: Vars) (Hnptr_x: ¬point x) (Hnaddr_x: ¬addressable x)
         ( z: Z) c:
155    AbstractSemantics (C, P) (X phiZ_to_Varz Hnptr_x Hnaddr_x z) c
          ( set_add Constraint_eq_dec (constraint1 Eq x z)
            ( set_add Constraint_eq_dec c (reset C x)), P)
     | asem_phiVarz_to_Varz CP
         ( x: Vars) (Hnptr_x: ¬point x) (Hnaddr_x: ¬addressable x)
160      ( v: Vars) (Hnptr_v: ¬point v) c: AbstractSemantics (C, P) (X phiVarz_to_Varz Hnptr_x Hnaddr_x Hnptr_v v) c

          ( set_add Constraint_eq_dec (constraint2 x Eq v)
            ( set_add Constraint_eq_dec c (reset C x)), P)
     | asem_phiOther_to_Varz CP
165    ( x: Vars) (Hnptr_x: ¬point x) (Hnaddr_x: ¬addressable x)
       ( e: OtherExprs) c: AbstractSemantics (C, P) (X phiOther_to_Varz Hnptr_x Hnaddr_x e) c

          ( set_add Constraint_eq_dec c (reset x C), P)
     | asem_phiPtr_to_Ptr CP
170    ( p: Vars) (Hptr_p: point p) (Hnaddr_p: ¬addressable p)
       ( v Vars) (Hptr_v: point v) c Hconsistency: AbstractSemantics (C,
       P)
         ( phiPtr_to_Ptr p Hptr_p Hnaddr_p v Hptr_v Hconsistency) c
```

```
        ( set_add Constraint_eq_dec (Eq constraint2 p v)
175       ( set_add Constraint_eq_dec c (reset C p)),
          ( update_with_existing_element P p Hptr_p Hptr_v Hconsistency v))
    | asem_phiAddr_to_Ptr CP
      ( p: Vars) (Hptr_p: point p) (Hnaddr_p: ¬addressable p)
      ( there Vars) (Haddr_y: addressable y) c Hconsistency:
180   AbstractSemantics (C, P)
        ( phiAddr_to_Ptr p Hptr_p Hnaddr_p there Haddr_y Hconsistency) c
          ( set_add Constraint_eq_dec c (reset C p),
            ( update_with_address P p Hptr_p Haddr_y Hconsistency y))
    | asem_phiOther_to_Ptr CP
185   ( p: Vars) (Hptr_p: point p) (Hnaddr_p: ¬addressable p)
      ( e: OtherExprs) c: AbstractSemantics (C, P) (P phiOther_to_Ptr Hptr_p Hnaddr_p e) c

        ( set_add Constraint_eq_dec c (reset C p),
          ( update_with_top Hptr_p P p))
190
    | asem_callVarz CP (x: Vars) (Hnptr_x: ¬point x) c:
      AbstractSemantics (C, P) (CallVars x) c
        ( set_add Constraint_eq_dec c (reset (reset_addr C) x) update_all_addr_with_top
          P)
195 | asem_callPtr CP (p Vars) (Hptr_p: point p) c:
      AbstractSemantics (C, P) (CallVars p) c
        ( set_add Constraint_eq_dec c (reset (reset_addr C) p) update_all_addr_with_top
          (update_with_top Hptr_p P p))
    | asem_callIgnoredVars CP (x: IgnoredVars) c:
200   AbstractSemantics (C, P) (CallOther x) c
        ( set_add Constraint_eq_dec c (reset_addr C)
          update_all_addr_with_top P)
    | asem_callNoVars CP c:
      AbstractSemantics (C, P) call c
205     ( set_add Constraint_eq_dec c (reset_addr C)
          update_all_addr_with_top P)
    | asem_join CP c:
      AbstractSemantics (C, P) join c
        ( set_add Constraint_eq_dec c C, P).
```

We naturally extends to these semantic paths. We write for example:

$$\theta \rightarrow^{*p} \theta'$$

with $p \in PATHS_{flows}$ to mean? er there $v_1\ v_2 \ldots, v_{not} \in V$, $c_1\ c_2 \ldots, c_{not} \in C$ $\theta_1\ \theta_2 \ldots, \theta_{not-1} \in K$ such as p is the path ($v_1\ c_1\ v_2\ c_2 \ldots, v_{not-1}\ c_{not}, v_{not}$) and

$$\theta\ v_1 \vdash c_1 \rightarrow \theta_1 \quad \xrightarrow{v_2\ c_2} \theta_2 \ldots \theta_{not-1} \quad \xrightarrow{v_{not}, c_{not}} \theta.$$

In Coq, a path is a list of pairs < node, constraint > the top of the list is the last node. To facilitate evidence, we de? Ne way of arti? Cial semantics of the empty road. We also note that unlike our de? Nition "paper", in Coq, a path ends with a bow without destination, surprisingly enough. This is however not a limitation because we can always consider the existence of a node V_{join} terminal: the nodes of this set are changing neither the abstract semantics nor the concrete semantics.

```
Definition Path: Type : = List (Nodes * Constraint).
inductive AbstractSemanticsPath: AbstractConfiguration → Path →
      AbstractConfiguration → Prop : =
 | AbstractSemanticsNil (k AbstractConfiguration)
   AbstractSemanticsPath nil k k
 | AbstractSemanticsCons (kk ' k '': AbstractConfiguration)
    ( p ': Path) (H: AbstractSemanticsPath kp ' k ') ( v: Nodes) (C: Constraint) (Hlast: AbstractSemantics
    k ' VCK ''):
   AbstractSemanticsPath k ((v, v) :: p ') k ''.

inductive ConcreteSemanticsPath: MemoryState → Path → MemoryState → Prop
      : = | ConcreteSemanticsNil (theta: MemoryState)

   ConcreteSemanticsPath nil theta theta
 | ConcreteSemanticsCons (theta theta ' theta '': MemoryState)
    ( p ': Path) (H: ConcreteSemanticsPath theta p ' theta ')
    ( v: Nodes) (C: Constraint) (Hlast: ConcreteSemantics theta ' vc theta ''):
   ConcreteSemanticsPath theta ((v, v) :: p ') theta ''.
```

We ask here a technical lemma? Rm us that the de transition relation ning abstract semantics is full left, that is to say there is a rule for each type of node.

Property 2 (Completeness of the left).

$$\forall v \forall \in V c \forall \in C k \exists \in K k' \in K k_{v,c} \quad -k'$$

Demonstration. We proved this lemma with Rooster, by case analysis on all types of nodes.

```
Theorem abstract_semantics_is_left_total:
 ∀CP n, ∃C ' P ' AbstractSemantics (C, P) n (C ' P ').
```

□

5.4.4 Conclude a path is not

We de? Ne a satisfiability relation between designed concrete and abstract configurations. Intuitively, designed concrete configuration satisfies a designed abstract configuration if every constraint of the designed abstract configuration is veri? Ed by the varying valuation in the designed concrete configuration. ? Conversely, a con abstract configuration is unsatisfiable if no con concrete configuration does satisfy; typically because it contains incompatible constraints between them. A path is impossible if there is no way to produce a designed satisfiable abstract configuration starting from a designed configuration "empty", that is to say a designed configuration without any constraint satisfied by all designed concrete configurations. In e? And, according to our de? Nition,

Disappointed nition11 (Satisfiability). A designed concrete configuration $\theta = (\sigma, \gamma, \alpha) \in \Theta$ satisfies a designed abstract configuration $k = (C, P) \in K$, what is written $\theta\ k$ if and only if,

$$\theta\ C$$
$$\wedge\ \forall p \in Vars_{ptr} P(p)\ \theta = > = \Rightarrow \alpha(P(p)) = \theta(p)$$

The designed configuration k is unsatisfiable, what one notes $1\ k$ if

$$\forall \theta \in \Theta\ \theta\ 1\ k$$

In Coq, we de? Ne the satisfiability in the same way.

```
Definition satisfiability_set_constraints theta C: =
  forall c, c set_In C -> satisfiability theta c.

Definition satisfiability_pointer_map theta (P: sig
    PointerMap_is_consistent): =
  forall p Hptr_p v Haddr_v (hprt (proj1_sig P) = p Hptr_p
    ptvar Haddr_v v)
  Valuation theta p = inr (alpha v Haddr_v).

Definition satisfiability_abstract_configuration theta k =
  satisfiability_set_constraints theta (fst k) / \
  satisfiability_pointer_map theta (snd k).
```

Correction of static analysis

The latter disappointed definition allows us to express the correctness of our static analysis. Assuming that the properties that we have assumed the concrete semantics are veri? Ed, we can prove the following proposition.

Proposition 1 (Correction of static analysis). *The abstract semantics maintains satisfiability vis-à-vis the designed concrete corresponding configuration.*

$$\forall p \in PATHs_{flows} \forall \theta_1\ \theta_2 \in \Theta\ \forall k_1\ k_2 \in K$$
$$(\theta_1 \rightarrow {}^{\bullet}_{p}\ \theta_2 \wedge k_1 \quad {}^{\bullet}_{p}\ k_2 \wedge \theta_1\ k_1) = \Rightarrow \theta_2\ k_2$$

Intuitively, this proposal says that if a way is possible, that is to say, its concrete semantics is de? Ned, while its abstract semantic produced a designed configuration compatible with the designed concrete resulting configuration.

We demonstrate this proposition by induction, demonstrating first the base case in the lemma below.

Lemma 1 (Correction for a transition). $\forall \theta_1\ \theta_2 \in \Theta\ \forall k_1\ k_2\ \forall \in K\ v\ \forall \in V\ c \in C\ (\theta_1$
$\overset{v,\ c}{-\!\!\rightarrow}\ \theta_2 \wedge k_1 \quad \overset{v,\ c}{-}\ k_2 \wedge \theta_1\ k_1) \Rightarrow \theta_2\ k_2$

Demonstration. This evidence was also carried out in Coq, by induction on the de? Nition of static analysis (which has already proved that it covered all types of nodes).

```
Lemma restricted_soundness: ∀ (v: Nodes) (c: Constraint)
  ( kk ': AbstractConfiguration) (theta theta ': MemoryState)
  ( AbstractSemantics kvck ') → ( ConcreteSemantics vc theta theta ') →
  satisfiability_abstract_configuration theta k →
  satisfiability_abstract_configuration theta ' k .
```

□

We can now prove the proposition 1 .

Demonstration. We make a simple induction on the length of *p,* using lemma 1 .

```
Theorem soundness: ∀ ( p: Path)
  ( kk ': AbstractConfiguration) (theta theta ': MemoryState)
  ( AbstractSemanticsPath kpk ') → ( ConcreteSemanticsPath theta theta p ') →
  satisfiability_abstract_configuration theta k →
  satisfiability_abstract_configuration theta ' k .
```

□

Prove the impossibility of a path

To prove that a way is impossible, we can:

1. Start with an empty con abstract configuration;
2. progress along the way by changing the designed configuration according to the rules of semantics presented in section 5.4.3 ;
3. stop when the configuration is unsatisfiable and declare the impos- sible way;

4. or achieve? N the way and declare the possible path. Formally, we de? Ne the set I impossible roads, and the whole E

executable paths, as follows:

De? Nition 12 (impossible roads and executable paths).

$$I = \{ p \in PATHs \mid \forall k_1\ k_2 \in K\ k_1 \qquad \bullet_p k_2 = \Rightarrow 1\ k_2 \}$$

$$E = \{ p \in PATHs \mid \exists\ \theta_1\ \theta_2 \in \Theta\ \theta_1 \rightarrow \bullet \theta_2 \}$$

the proposition is established.

Proposal 2 (Paths detected as impossible really are). *If a path is detected as possible by static analysis, so no concrete enforcement can not borrow it.*

$$E \cap I = \varnothing$$

Demonstration. Suppose there $p \in I \cap E$. As $p \in E$, it exists $\theta_1\ \theta_2 \in \Theta$ such as $\theta_1 \rightarrow \bullet_p \theta_2$. Let then $k_1 \in K$ a designed abstract configuration as $\theta_1\ k_1$ (such designed configuration still exists because it known? t consider (*C, P)* or *= C {* true} and *P = x → 7>* which clearly satisfies this property). As is a total relationship

left, so there is a designed abstract configuration $k_2 \in K$ as k_1 $*_p k_2$. By
Therefore, the proposed correction established above, we θ_2 k_2. However, as $p \in$ I it was assumed 6
k_2 which gives a contradiction. □

Finally, according to this proposal, if it is shown that P $\subseteq$ I then we prove
P $\cap$ E = $\varnothing$. The paths that contain instructions generating a greeting information while dodging the hooks LSM (all P) not be borrowed during actual performances. Therefore they do not allow to dodge the control of information flow.

5.4.5 Management of loops

Proposal 1 established in the previous section shows that we can consider all the paths of length? ned. The roads in? Nite are not relevant in our study since a system call never ending execution stands for kernel programming error and is of no interest in terms of control of greeting.

However, if the graph contains cycles, corresponding to loops in the code, all P paths arbitrarily large length can be in? nite. Fortunately, the compiler applies few restrictions on representation of cycles in the graphs. First, all the loops have a unique loopback node that is a joining node having exactly two arcs: one from before the loop and from the interior thereof (thus forming a ring in the graph). Ordinarily, it is the input node in the loop, but it is possible C to jump within a loop and these cases also exist in Gimple. On the other hand, two nested loops are always di? Érenciées, they never share the same loopback node. We assume that these properties are veri? Ed. In practice, they are guaranteed by CCG.

De? Nition 13 (Loop). a loop l is de ned by the data?:

- $V_l = \{v_l, v_1, v_2 \ldots, v_{not}\} \subseteq V$;
- $v_l \in V_l \cap V_{join}$;
- and $T_l = (v_l, c_1, v_1), (v_1, c_2, v_2) \ldots (v_{not}, c_{not}, v_l) \in E +$ in the graph. The following properties are observed:

- v_l is the unique node V_l to have a predecessor in $V \mid V_L$ It is named loopback node of the loop.

- v_l is the loopback node of any other loop of the graph. We de? Ne an equivalence relation between paths P. We ask that two paths are equivalent if they are identical to cycles close, that is to say if removing all cycles of the two paths gives the same acyclic way.

De? Nition 14 (normal form of a path and equivalence relation). Either way
$p \in PATHs$, the normal form of p denoted ˆ p is the way P where all cycles, that is
ie sub-paths from one node $v \in V$ at the same node v are deleted.

They say two paths p_1 $p_2 \in PATHs$ they are equivalent, which is denoted
$p_1 \equiv p_2$ if and only if, ˆ $p_1 =$ ˆ p_2.

It's clear that $\equiv$ is e? ectively an equivalence relation because equality is a re relationship? reflexive, symmetric, and transitive. We partition *P ATHs* according to this equivalence relation. easy to show that there is a unique normal form in each equivalence class.

Designed configuration outcome loops

Each loop, we calculate a designed abstract configuration that is independent of the number of iterations by removing the con? Guration start loop all constraints on variables change? Ed inside the loop (this can match all the variables $V\,ars_{same}$ if a function call is present among others). We call this con? Guration "con? Guration result of the loop."

De? Nition 15 (Designed configuration result of a loop). Be a loop $l = (V_l\, v_l\, T_l)$ as $T_l = (v_l\, c_1\, v_1), (v_1\, c_2\, v_2) \ldots (v_{not}, c_{not}, v_l)$ and designed configuration $k \in K$. The designed configuration result l from k is noted k_l. It is calculated by removing k all constraints on variables change? ed inside the loop.

Formally, we de? Ne a function purge: $K \times V \to K$

- $v \in V_{assign}$ of shape $x = e$ with $x \in V\,ars_Z$

$$purge((C, P), v) = (reset(C, x), P)$$

- $v \in V_{assign}$ of shape $x = e$ with $x \in V\,ars_{ptr}$

$$purge((C, P), v) = (reset(C, x), P[x \leftarrow\!\!\!\!\rightarrow])$$

- $v \in V_{assign}$ of shape $x = e$ with $x \in V\,ars$

$$purge((C, P), v) = (C, P)$$

- $v \in V_{same}$ of shape $*p = e$ with $P(p) \in V\,ars_Z$

$$purge((C, P), v) = (reset(C, P(p)), P)$$

- $v \in V_{same}$ of shape $*p = e$ with $P(p) \in V\,ars_{ptr}$

$$purge((C, P), v) = (reset(C, P(p)), P[p \leftarrow\!\!\!\!\rightarrow])$$

- $v \in V_{same}$ of shape $*p = e$ with $P(p) = >$

$$purge((C, P), v) = (reset(C\,O(p)), P[q \leftarrow\!\!\!\!\rightarrow \mid O(P) \cap V\,ars_{ptr}])$$

- $v \in V_{\varphi}$ of shape $x = IHP < e_1 \ldots, e_{not} >$ with $x \in V\,ars_Z$

$$purge((C, P), v) = (reset(C, x), P)$$

- $v \in V_{\varphi}$ of shape $x = \text{IHP} < e_1 \ldots, e_{not} >$ with $x \in Vars_{ptr}$

$$\text{purge}((C, P), v) = (\text{reset}(C, x), P[x \leftrightarrow])$$

- $v \in V_{call}$ of shape $f()$

$$\text{purge}((C, P), v) = (\text{reset}(C\, Vars_{same}), P[q \leftrightarrow | q \in Vars_{same} \cap Vars_{ptr}])$$

- $v \in V_{call}$ of shape $f(e_1 \ldots, e_{not})$

$$\text{purge}((C, P), v) = (\text{reset}(C\, Vars_{same}), P[q \leftrightarrow | q \in Vars_{same} \cap Vars_{ptr}])$$

- $v \in V_{call}$ of shape $x = f()$ with $x \in Vars$

$$\text{purge}((C, P), v) = (\text{reset}(C\, Vars_{same} \cup \{x\}), P[q \leftrightarrow | q \in (Vars_{same} \cap Vars_{ptr}) \cup \{x\}])$$

- $v \in V_{call}$ of shape $x = f(e_1 \ldots, e_{not})$

$$\text{purge}((C, P), v) = (\text{reset}(C\, Vars_{same}), P[q \leftrightarrow | q \in (Vars_{same} \cap Vars_{ptr}) \cup \{x\}])$$

The designed configuration result of the loop l from the designed configuration k? Is de ned as:

$$k_l = \text{purging}(\text{purging}(\ldots(\text{purge}(k, v_{not}), \ldots) v_1), v_l)$$

We show that this approach is correct by showing that designed abstract obtained configuration is satisfied by at least the same con? Gurations that any concrete designed configuration that would be obtained by applying the rules of the abstract semantics along a path of the loop l (that is to say, a path made up of a number of iterations but arbitrarily large? or cycle component

l). The proof is based on an order relationship between the designed abstract configurations and ownership that this order is compatible with satisfiability.

De? Nition 16 (Order relation on designed abstract configurations). It disappointed nes order relationship $\subseteq K \times K$? On con gurations abstract by:

$$(C, P) \quad (C', P') \Leftrightarrow C \subseteq C' \wedge (\forall p \in Vars_{ptr}\, P(p) = > \vee P(p) = P'(p))$$

Proposal 3 (Compatibility with).

$$\forall \theta \in \Theta\ \forall k, k' \in K\ kk' \Rightarrow (\theta k' \Rightarrow \theta k)$$

Demonstration. Suppose we have k $k' \theta$ k' but $\theta 1$ k. So asking $\theta = (\Gamma, \sigma, \alpha)$, $k = (C, P)$ and $k' = (C', P')$, we know that

$$(\exists c \in \theta\, C 2\, c) \vee (\exists p \in Vars_{ptr}\, P(p)\, 6 = > \wedge \alpha(p)\, 6 = \theta(p))$$

For analysis of the case:

- $\exists c \in \theta C 2 c$. Yes $c \in C$ so $c \in C'$ because $C \subseteq C'$ by hypothesis. But, as we know $\theta C'$ so θc, hence a contradiction.
- $\exists p \in Vars_{ptr} P(p) 6 = > \wedge \alpha(x) 6 = \theta(p)$. Yes $P(p) 6 = >$ so $P \gamma p) = P(p)$ by hypothesis. But we know that $\forall p \in Vars_{ptr} P \gamma p) 6 = > \Rightarrow \alpha(p) = \theta(p)$, so we $P(p) 6 = > \wedge \alpha(p) = \theta(p)$, contradicts the assumptions.

□

We prove now a lemma stating that the designed configuration result of the loop l is smaller than any designed abstract configuration computed from a path l.

Lemma 2 (The designed configuration result is smaller than any con? Guration calculated on a path l). *Either a loop* $l = (V_l v_l T_l)$.

$$\forall (v, c, v_\gamma) \in T_l \forall k, k' \in K k_{v,c} \quad - k' \Rightarrow k_l \quad k'$$

Demonstration. We think that $k = (C, P)$, $k_l = (C_l P_l)$ and $k' = (C' P')$. One must then prove:

1. $C_l \subseteq C'$
2. $\forall p \in Vars_{ptr} P_l(p) = > \vee P_l(p) = P \gamma p)$

1. $C \subseteq C$. By case analysis to the nature of v and the de? nition of the rules semantic, it is clear that the only constraint C who are not in C' Constraints are removed by the function reset. These constraints therefore cover the variable assigned in a node has? Assignment, or the variable $P(p)$ or a variable $O(p)$ in the case of a? assignment through a pointer p, or $Vars_{same}$ in the case of a function call. For the de? Nition of k_l these constraints C not part of C_L As more $C_l \subseteq C$ it is concluded that $C_l \subseteq C$.

2. $\forall p \in Vars_{ptr} P_l(p) = > \vee P_l(p) = P \gamma p)$ All pointers p such as $P(p) 6 = > \wedge P \gamma p) 6 = P(p)$ are pointers assigned directly or pointers in $Vars_{same}$ assigned through a pointer. In any case, by de? Nition k_l we have $P_l(p) = >$ So we can conclude that $\forall p \in Vars_{ptr} P_l(p) = > \vee P_l(p) = P \gamma p)$.

□

We can now prove a version of the previous lemma extended to an entire way.

Proposal 4. *Be a loop* $l = (V_l v_l T_l)$ *and p a path* T_L. *We have :*

$$\forall k, k' \forall \in K \theta \in \Theta (k *_p k' \wedge \theta k \gamma \Rightarrow \theta k_l$$

Demonstration. As is an order relation, it is transitive and by induction on the length of the path p, we can conclude from Lemma 2 than $k *_p k' \Rightarrow$ $k_l \quad k$. By proposition 3 , So we have $\forall \theta \in \Theta \theta k' \Rightarrow \theta k_L$. □

Conclusion on path analysis

According to the latest proposal he know? T analyze the normal form of each equivalence class. All normal forms is the set (? Or) acyclic graphs paths. For analysis, we restrict ourselves to calcu- late the subsets of acyclic paths P. We show that if the normal form corresponds to an impossible way then all the paths of all is impossible. We then analyze the normal forms and thus conclude on the impossibility of each of the paths P.

Consider $S \in P \setminus =$ a coset according $\equiv$ and p a path S. We want to prove that ^

$p \in$ I involved $p \in$ I.

We show first that the rules of abstract semantics preserves the order on designed abstract configurations.

Lemma 3 (Preservation).

$$\forall k_1\, k'_1\, k_2\, k'_2 \in K\, v \in V\, c \in C \left(\left\{ \begin{array}{l} k_1 \xrightarrow{v,c} k_2 \\ k'_1 \xrightarrow{v,c} k'_2 \end{array} \right. \right) \wedge k'_1 \equiv k_1 \Rightarrow k'_2 \equiv k_2$$

Demonstration. suppose that $k_1 = (C_1, P_1)$, $k_2 = (C_2, P_2)$ $k'_1 = (C'_1, P'_1)$, $k'_2 = (C'_2, P'_2)$ and $k'_1 \equiv k_1$.

Is $q \in Vars_{ptr}$ a pointer. ? By the de nition of semantic rules, we least one of the following apply:

1. Case $\begin{cases} P'_2(q) = P'_1(q) \\ P_2(q) = P_1(q) \end{cases}$.

This is the case (*v, c)* does not change the variable information which q point. As $k'_1 \equiv k_1$ we have $P'_1(q) =>$ or $P'_1(q) = P_1(q)$ Which give $P'_2(q) =>$ or $P'_2(q) = P_2(q)$.

2. Case $\begin{cases} P'_2(q) => \\ P_2(q) => \end{cases}$.

This is the case (*v, c)* exchange information about the variable that q tip>. In this case, it was trivially $P'_2(q) =>$ or $P'_2(q) = P_2(q)$.

3. Case $\exists q' \in Vars_{ptr}$ $\begin{cases} P'_2(q) = P'_1(q') \\ P_2(q) = P_1(q') \end{cases}$.

This is the case where the transition (*v, c)* exchange information about the variable that q points to the destination of another pointer q'. As $k'_1 \equiv k_1$ we at $P'_1(q') =>$ or $P'_1(q') = P_1(q')$, Which give $P'_2(q) =>$ or $P'_2(q) = P_2(q)$.

4. Case $\exists x \in Vars_{same}$ $\begin{cases} P'_2(q) = x \\ P_2(q) = x \end{cases}$.

This is the case where the transition (*v, c)* exchange information about the variable that q peak to a variable x. In this case, it was trivially $P'_2(q) =>$ or $P'_2(q) = P_2(q) = v$.

Is $C_r, C_{at} \subseteq C$ sets respectively deleted constraints and added to C_1 and C'_1 the transition (*v, c) (* according to the rules of abstract semantics). We have

$$\begin{cases} C_2 = (C_1 \setminus C_r) \cup C_{at} \\ C'_2 = (C'_1 \setminus C_r) \cup C_{at} \end{cases}$$

. This gives trivially $C'_1 \subseteq C'_2 = \Rightarrow C_1 \subseteq C_2$.

So we proved that $C'_2 \subseteq C_2$ and $\forall q \in Vars_{ptr}(P'_2)(q) = \nu P'_2(q) = \nu P_2(q)$), which is to write $k'_2 \; k_2$. □

We extend the previous lemma to the paths.

Lemma 4 (order preservation along a path).

$$\forall k_1\, k'_1\, k_2\, k'_2 \in K\, p \in P \quad \left(\begin{array}{c} k_1 \;\; *_p k_2 \\ k'_1 \;\; *_p k'_2 \end{array}\right) \wedge k'_1 \;\; k_1 \Rightarrow k'_2 \;\; k_2$$

Demonstration. By induction on the length of the path p (a path length $n + 1$ is a path of length *not* followed by a last transition (*v, c)).* As is an order relation, is transitive, which gives us the result trivially. □

lemma 2 P. 120 gives a similar result about loops. We can finally prove the following proposition?:

Proposal 5 (Analyze the normal form of a known equivalence class? T conclude on the impossibility of the whole class). *Given $S \subseteq P$ a coset according $\equiv$. and $[s] \in S$ the normal form of S we have $[s] \in I \Rightarrow S \subseteq I$.*

Demonstration. Consider a path $p \in S$. By de? Nition ˆ $p = [s]$. suppose that $[s] \in I$ but $p / \in I$. We can then assume that there $k_1\, k_2 \in K\, k_1 \; *_p k_2$ such as k_2. p is a composition of sub-paths, some of the cycles. By de? Nition $[s]$ is the same path without cycles. By induction on the length of the sequence of sub-paths p and Lemma 4 for sub-paths are not loops, and Lemma 2 for sub-paths are, and transitive , he exist $k'_2 \in K$ such as $k_1 \; *_{[s]} k'_2$ and $k'_2 \; k_2$. By proposition 3 It follows $[s]$, in contradicts the hypothesis. □

5.5 Implementation

Since version 4.3, the compiler GCC supports the addition of compilation passes additional tion in the processing chain in the form of gre? ons that can be dynamically loaded. The gre? Ons loaded can bene? T from all internal representations and all the functions implemented by GCC. Normally these gre? Ons are used to add enhancements but they can also be used for static analysis implementations. Some gre? Ons have in particular been integrated with the kernel sources in the latest versions.

We implemented two gre ons to perform static analysis in this section?: *Kayrebt :: Callgraphs,* already presented in Chapter 4 and *Kay- REBT :: PathExaminer2*. The first is very simple and only allows to extract the code of a kernel function calls graph. The call graph is a database indicating for each function, what are the functions that are called in the code, regardless of the conditions or the order of the calls. The call graph has allowed us to identify? Er automatically what hooks LSM

reachable from every system call, which allowed us to properly force their *inlining.*

The gre? We *Kayrebt :: PathExaminer2* is the one who actually implements the analysis. We do not extract the code graphs like those shown in? Gure 5.1 ; in fact, we are working directly on the internal representation GCC graphs

of? control ots. This prevents a transformation that could be a source of interpretation of the code errors. Moreover, one can thus bene? T from other compiler data structures like the alias oracle. The analysis is done independently on each system call and within each system call on each instruction generating a greeting. The points where vows are generated must be annotated by hand with a function call pseudo-managed by our gre lesson. All intermediate functions between the system call and functions where the hooks are located LSM and generation of points of vows must be marked by a specific attribute? that to GCC a? n to force their *inlining*.

Running the analysis starts at the entrance to the system call. The code essentially just e? Ectuer a course in? Graph width control ot a? N to explore all paths methodically. As and extent of exploration, one designed abstract configuration, initially empty, is updated with the constraints of nodes and arcs. Each modi? Cation of the con? Guration, its satisfiability is veri? Ed. We use Yices [23], An SMT-solver with the arithmetic theory, to con? Er satisfiability of constraint sets. If the designed configuration is unsatisfiable, then the path is abandoned. When a conditional branch is encountered, with several outgoing arcs, the designed configuration is duplicated and the analysis continues independently on each of the paths. If an analysis continues until the? N of the path (the return of function) with a designed satisfiable configuration, then this path can not be proven impossible. It has? Ket on the output of the tool to allow veri? Cation further manual.

5.6 Results

We originally developed the analysis kernel o? Heaven 4.3, compiled for the x86_64 architecture with the default con? Guration options, but we subsequently reproduced on the kernel 4.7. In reality, the analysis is independent of the architecture and kernel version but it must be conducted independently in each case in order to draw useful conclusions. In e? And, depending on the architecture, version and con options? Configurations, system calls and the means to produce greeting information vary.

We cloned the code repository o? Linux kernel sky and we have prepared a branch for each system call in which we placed the annotations to start the analysis. The results thereof are listed in Table 5.2 .

In most cases, the result of the analysis is clear: either there is no way not passing through a hook LSM Or all of these paths are impossible. In all these cases, it can be concluded that the hooks LSM are placed appropriately.

Some system calls that we have identified? Ed as causing greeting information do not have hooks LSM . This is the case of tee, causing a flow tube to tube information, as well as mq_timedreceive and mq_timedsend respectively recovers and inserts messages in? the POSIX message.

5.6.1 mq_timedsend and mq_timedreceive

Unlike the corresponding system call? System V messages msgsnd and msgrcv, the calls mq_timedsend and mq_timedreceive have no cro

Table 5.2 - Results of static analysis

system call	result	Details
discrete flow		
read	X	All paths P are impossible
readv	X	All paths P are impossible
preadv	X	All paths P are impossible
pread64	X	All paths P are impossible
write	X	All paths P are impossible
writev	X	All paths P are impossible
pwritev	X	All paths P are impossible
pwrite64	X	All paths P are impossible
sendfile	X	All paths P are impossible
sendfile64	X	All paths P are impossible
splice	~	No hook for greeting tube tube
		All other roads are impossible
tee	✗	No hook LSM
vmsplice	~	A path is possible
recv	X	All P is empty
recvmsg	X	All P is empty
recvmmsg	~	A path is possible
recvfrom	X	All P is empty
send	X	All P is empty
sendmsg	X	All P is empty
sendmmsg	~	A path is possible
sendto	X	All P is empty
process_vm_readv	X	Paths are possible but see below
process_vm_writev	X	Paths are possible but see below
migrate_pages	X	All P is empty
move_pages	X	All P is empty
fork	X	All P is empty
vfork	X	All P is empty
clone	X	All P is empty
execve	X	All P is empty
execveat	X	All P is empty
msgrcv	X	All paths P are impossible
msgsnd	X	All P is empty
mq_timedreceive	✗	No hook LSM
mq_timedsend	✗	No hook LSM

Continued from Table 5.2. Results of static analysis

continuous flow		
shmat.	X	**All P is empty**
mmap_pgoff.	X	**All P is empty**
mmap.	X	**All P is empty**
ptrace.	X	Paths are possible

waste LSM . We interviewed the developers of the modules LSM on their list of di? usion on the reasons for the absence. Stephen Smalley, developer and maintainer of SELinux, provided the following explanation of **elements [84]:**

- **These system calls have been added to the core after the *framework* LSM They have therefore not been** considered in the original design.
- Unlike the System V interfaces, POSIX interfaces are developed based on the system? Les. The? The POSIX message bene? T from security mechanisms through this. In particular, open a? The **POSIX message needs to go through the hook LSM Opening files?.**
- **brackets LSM placed in the system calls read and write have essentially been placed for the** revalidation of access permissions to? le for each reading and writing. Revalidation is considered relevant especially for the? Le standard and not other types of live objects in the system? Les. **Therefore, the hook LSM for opening? the messages is considered out? health.**

These reasons underlie in fact that the implementation of control? Ow of information is not an intended use **in *design* of LSM .**

5.6.2 tee, splice and vmsplice

The system calls tee, splice and vmsplice are conceptually transact read and write could do. These calls take advantage of the implementation of the tubes in the Linux kernel to make copies of information between pipes or between a tube and a? Standard file without actually make copies byte to byte in memory. The reasons for which some missing hooks similar to those advanced for? The POSIX message. The tubes are considered as not being subject to revalidation. Therefore, when a system call is used only for reading or **writing in tubes, brackets LSM have been omitted. One can also see a limitation of our static analysis here. We have veri? Ed the presence of hooks LSM in system calls but we have not veri? ed if the hooks are suitable for observing the flow? caused by system calls. For example, when splice ? Is used to cause a flow** of a tube to a file, a single hook is crossed: the hook normally used by the security modules to con st write permission in the shit?. The greeting information monitor could therefore see a mistake and greeting the calling process to the? Le, instead of the tube to the? Le. Correct monitor implementation therefore requires a means of distinguishing the di? Erent uses of the same hook from the viewpoint of monitoring greeting.

5.6.3 process_vm_readv, process_vm_writev, ptrace

These three system calls used to make greeting memory memory between two processes. At first glance, some paths of executions are not covered. However, manual analysis led us to first identify these paths as corresponding to a particular situation: a reading process its own memory (that is to say a *thread* reading the memory of a *thread* within a single process). This does not correspond to a greeting of information after we die? Nition because *threads*

share anyway their memory, which forms a single information container.

5.7 Conclusion

The work presented in this chapter was presented at the formal conference in 2017 [36]. We listed system calls that produce greeting and we studied all execution paths that generate them. We can conclude that LSM is generally well suited to monitoring? ow of information, with a few exceptions that our approach can correct. Based on our results, we produced *patches* Linux kernel correcting the problems we have identified? ed and we have revalidated the new kernel by repeating the analysis on the framework LSM

completed. However, work is still needed for further analysis. In e? And we have veri? Ed that the execution paths producing vows are well covered by all appropriate hooks. The case of splice is problematic for example even if the flow is between a file and a tube, the single hook? LSM

this is file_permission that normally characterize a greeting of? le towards the calling process. This case is unique to our knowledge, so we can treat it easily, but a clear semantics of system calls in terms of greeting information is needed to make sure.

The clear advantage of our approach is that we do not need to formalize a complete semantics for the dialect of the C language (including extensions implemented only by GCC) Used for kernel development. On the other hand, be integrated toolchain allows us to bene? T from a lot of internal data structures such as graphs? Ots control, analysis of loops or the alias oracle. Moreover, as these structures are those generated by GCC The code that is scanned is perhaps not one that is written, and may not match the intent of the programmer, but it is the code that will be executed, or at least a more representation close to this code than the original one. Although *frameworks* analysis as BLAST [4] Are much more comprehensive and powerful, they give semantics to the C language can di? Anaging that of

GCC and they fail to bene? t from the internal data structures of the compiler. In? N, using a gre? We GCC allows us to insert ourselves in any place of the toolchain. In particular, fit relatively late can analyze a pre-code worked, longer but with fewer types of instructions di? Erent, and where all connections and loops are transformed into type instructions *goto.* The major disadvantage of the use of gre? Ons is that as the 'E compilation? Ectue? Le by? Le, the possibilities of static translation of trans-analysis units are limited. Also, debug analysis returns to debug GCC whole, which is fairly di? cult. The use of static analysis "assisted by the compiler" seems a promising approach to develop quickly and safely analyzes of modest size, with limited objectives.

chapter 6

Rfblare: an implementation of Blare able to manage the competition between system calls and memory screenings

In the previous chapter, we identi ed a necessary condition for the correct implementation of monitors flow information:? The presence of a hook LSM in each system call causing a greeting information. This condition ensures that it is possible for a monitor with implemented LSM observe all vows direct. However, this condition is not *know? Health* to ensure the observa- tion of all ow?. In e? And our experiments implementations flow of information led us to identi? Er situations where, although all individual flow of information are observed, the greeting indirect escape the vigilance of the monitor of? ow of information. These cases have proved symptomatic of

competitive conditions calls between systems operating in the same information containers. To illustrate this problem, we can consider the situation presented in Figure 6.1 . The process *cat* a copy *take a dump* in *tube (in the command line, the tube is anonymous and created by the use of "| "), The process wc -l* then reads to calculate the number of lines of text. The process *wc -l* a? che the result to standard output

/ Dev / stdout. Three containers are initially marked with a symbol, also called *color* indicating the class of information stored in the container: *the* with *?, cat* with *†, wc -l* with *•*. • The task of monitor information flow is to propagate brands from one container to another in a stream to save the di usion of information?; this is called the *propagation of hues*. If all the vows were observed in the order they are e? Ectués, the monitor should copy the mark

? of *take a dump* at *cat,* then the brand *†?* of *cat* at *tube,* then *wc -l* and? n, the brand *• †?* of *wc -l* at */ dev / stdout*. However, system calls, the point where the greeting is observed, that is to say, the hook LSM And the point where the stream? E really? Ectué (typically a function call later in the call) are not executed atomically.

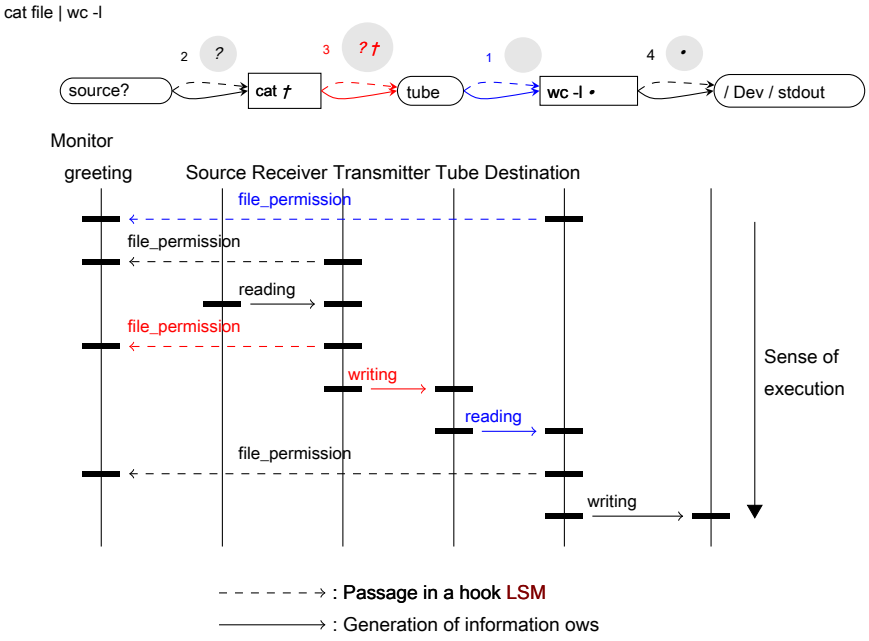


Figure 6.1 - Example of execution problems

It is therefore quite possible to observe a sequence of operations such as the one detailed in the table of the? Gure 6.1 . The execution of greeting and observation are in order di? Erent, therefore, the spread is incorrect. The first diagram shows the di? Erence between the result of the spread of colors and di? Usion of information. The? Dotted arrows represent the spread, e? Ectuée in the order indicated by the numbers,? Solid arrows indicate the? Ow e? Ectués, in order from left to right. We note that although the information is transferred from the? Le *source* towards *destination,* the colors are not propagated accordingly. Normally, / dev / stdout should receive the shades

? ƒ •; in reality, here it only gets • So the greeting indirect source and cat towards / Dev / stdout are? nally missing.

This problem occurs because of a *race condition* on the information container *tube.* As the sequence < observation of vows, generation of greeting > is not atomic, it is possible to observe two vows in a direct order di? erent of their generation, and therefore miss a? ux indirect. Although a monitor? Ow of information could tolerate a certain lack of precision, the problem of the conditions of competition arises more seriously considering the vows continuous. In e? And if two shared unemémoire processsus share, then it is essential to consider the greeting reaching each of them also reach the other. Imagine that a given process read from a? Le for example. He can choose as destination memory buffer of the system call read a portion of the shared memory. One must also consider that if two processes *AT*

and *B* share a share a memory area, and the process *B* and *C* share another second area, then the process *AT* and *C* share a memory area that, although none of the two processes has created explicitly.

We studied three monitors? Ow of information implemented with LSM :

Laminar [76] and *KBlare* [33], Developed for the generic Linux kernel, and *Weir* [67] Developed for the Android Linux kernel. In the first subsection, we give two examples of attacks exploiting the competitive environment and vows to escape the continuous monitors? Ow of information. Next, we describe a shade propagation model for formalize:

1. what would be an ideal spread, corresponding exactly to the stream made;
2. the spread of hues produced by the monitors of information ows studied;
3. di? Erence between the two spreads.

This formal model has allowed us to develop a solution for proper propagation, if not as accurate as the ideal spread, ie spread calculating a overapproximation shades containers greeting information when one does not know the exact order of flux?. This solution is based on a simple intuition. As it is impossible to match the observation of vows with their generation, we add the observation points *not* of ux. We consider that between the points where the start and? N of the vows are observed, it is *activated* and instead may have zero, one or more times, at any moment. We calculate the spread of hues in the light of all ow? *activated*

simultaneously. This solution allows us to manage both the competition concerns as well as the greeting continuing, by considering more the greeting as atomic events. We have demonstrated the correctness of our algorithm using the Coq proof assistant [93]. Throughout this chapter, we support the die elements? Definitions and theorems of their correspondence in Coq. The formal description of the algorithm and its proof are available in Annex C and on the project website at Blare https://blare-ids.org/rfblare/ . We show the implementability and usefulness of our new propagation algorithm *Rfblare,*

a new version of *KBlare* implementing only the spread of colors, minus the part of veri? cation of the legality of flux?. We show that *Rfblare* induces minimal performance overhead on actual tests.

6.1 Attacks on monitors? Ow of informa- tion implemented with LSM

6.1.1 Operation of a race condition between read and write

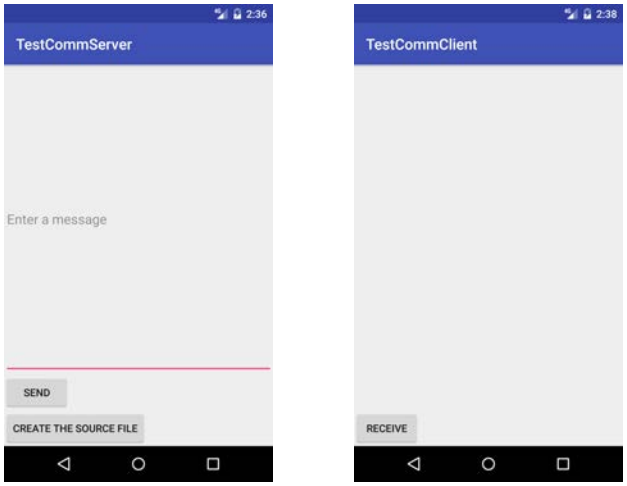
KBlare, Laminar and *Weir* use the hook file_permission to e? ectuer the spread of colors to or from the files?, during a read or a write. Two system calls read and write may compete if they operate on the same? le. This is true even on uniprocesseurs systems. In e? And system calls can release the CPU and sleep between passing in the hook and instruction causing the stream?. We tested a similar attack in the example presented in the introduction to this chapter:

```
mkfifo tube; cat <pipe> Destination & cat <source> tube
```

We create a named pipe *tube* Standing in the folder where the test takes place, a? n can monitor changes its hue. With this order, it is observed *KBlare*
is unable to propagate the colors correctly because he observes - in most executions, the scenario is not deterministic - playing the pipe, blocking before her writing. In fact, we discovered this problem conditions

competition in finding, in a unit test, that *KBlare* not always transferred the colors correctly when blockers containers writings-readings, such as pipes and network sockets.

To con? Rm that the problem is not related to *KBlare* but the implé- mented with monitors LSM in general, we countered the attack on *Weir*. This type of attacks on Android is more difficult to achieve because the processes are strictly isolated enough by default. We used a context of attack somewhat arti? Sky. We have developed two applications, a client and a server (corresponding to the reader and writer of the tube), presented in the? Gure 6.2 . The two applications are installed with the same identi? Ant user, so they share the same



(A) Tube Player (B) the tube Writer

Figure 6.2 - Applications developed to lead the attack on *Weir*

installation folder. To play the attack must manually create a tube in this case. At startup, the writer, the application *TestCommServer* request a shade system *Weir*. Subsequently, the user can write a message in a text box and press the button "Create the source? It." This button creates a? Source file, tinted to the value returned by *Weir* and containing the text entered in the dialog. Then, pressing the "Send" button copies the? Le in the tube. The tube reader application *TestCommClient,* has a text box and a button "Receive". Pressing this button activates the reading of the tube and has it? Chage of its contents into the text box, con? Rmed the greeting indirect information of the writer to the client via the tube. The spread of hues can be observed in the debug interface, observing the trace provided by *Weir* and transcribed with comments in the extract 6.1 . an interesting phéomène is observed: if you click on the receive button first, the writer's tone is not propagated to the reader; However, if you click on the transmit button before receiving button, the shades of pagation pro complies with vows made. This con? Rmed experimentally analyzing the? Gure 6.1 . The condition of competition here is particularly easy to start because the readings from the empty tubes are blocking until another process writes it. We have exploited this fact to facilitate attacks

```
1 === Create the source file by the issuer ===
2 <4> [3060.824300] WEIR_DEBUG: File Permissions. pid 11372,
        file /data/data/com.example.lgeorget.testcommclient/files / source

    <4> [3060.824525] WEIR_DEBUG: Label is pid 11372 is:
        96263937442022980 {,}

5 === Reading through the tube receiver ===
    <4> [3065.723624] WEIR_DEBUG: File Permissions. pid 11462,
        file /data/data/com.example.lgeorget.testcommclient/files / tube

    === no label on the tube or on the recipient here ===

    === Reading by the source transmitter ===
10 <4> [3068.259636] WEIR_DEBUG: File Permissions. pid 11372,
        file /data/data/com.example.lgeorget.testcommclient/files / source

    <4> [3068.260424] proc_label 96263937442022980 = {,} <4> [3068.261075] file_label
    96263937442022980 = {}

    === Script by the transmitter in the tube ===
15 <4> [3068.261268] WEIR_DEBUG: File Permissions. pid 11372,
        file /data/data/com.example.lgeorget.testcommclient/files / tube

    <4> [3068.261486] WEIR_DEBUG: Label is pid 11372 is: <4> [3068.261524] 96263937442022980 {}


    === Writing by the destination receiver ===
20 <4> [3068.266815] WEIR_DEBUG: File Permissions. pid 11462,
        file /data/data/com.example.lgeorget.testcommclient/files / destination

    <4> [3068.267042] WEIR_DEBUG: Label is pid 11462 is: {} === recipient must send the tag to the destination here

        but it has not gained ===
```

Extract 6.1 - Extract traces issued by *Weir* during the successful execution of the attack using *TestCommClient* and *TestCommServer*

but it is important to note that with a? le standard, the competitive environment also exist for *KBlare* like *Weir*.

In the case of *Laminar,* this attack is not directly possible because the devel- added loppeurs synchronization system calls *read* and *write,* so that the performance of each read or write operation is atomic. This solution prevents e? Ective any possible race condition but we do not consider however that it is satisfactory in all cases. On the one hand, it decreases performance by sacri? Ant parallelism very frequent calls. A reading from a system? Files with a certain lag, as a system of? Les on the network for example, block all other reading-writing. Secondly, it alters the semantics of certain information containers that are supposed to block the playback when empty, such as pipes and network sockets. This semantics is expected and exploited by many applications. This is not a problem for *Laminar* imposing in any way the use of non-blocking manner tubes to avoid a hidden channel? ow of information, but it is unacceptable for *KBlare* and *Weir* who do not want to require porting appli- cations. Moreover, if we extend this solution to all system calls that can compete, the risk of causing deadlocks becomes.

6.1.2 Operation of vows of continuous information

Read or Write? Le can be done with the family of system calls read and write but this is not the seuleméthode. It is possible for a process to project a? Le in his memory, that is to say, to ask the kernel via the system call mmap to match a certain portion of its address space to a range of the same length in the? le, so that access to these addresses are translated by the core to access? le underlying. This is very common; in fact, this is the way that the process load their executable memory. This mechanism also allows the establishment of shared memories. There are two interfaces for handling shared memories o ered processes: the old System V interface and the new POSIX interface. Sharing their internal implementation but use di? Erently. The shared memories System V have own system of collection calls and each have an identi? Single ant while POSIX memories are only an abstraction around the projections? Files in memory. There are no dedicated system calls to manipulate only the functions of the standard library. Internally, they realize with the projection mmap of a? le common processes wishing to share memory.

The vows continued not taken into account by *Weir* which focuses essentially on files? and processes. *Laminar* either does not offer solutions to this problem. However, we note that the problem is familiar to developers. In the source code *Laminar,* we find the following comment: "XXX: Should do something about the mmaped?. "[75 , ?take a dump security / difc.c, l. 944]. This suggests that the problem is considered important by developers but is not trivial to solve. *KBlare* offers some solutions to spread the colors around the projections and shared memories. First, when a process reads a? Le, the colors are propagated not only the process but also to files he plans to write permissions. It is also described in the thesis that Hauser *KBlare* manages shared memories of System V now to update the list of identi? ers shared memories associated with each process and using that list to spread the tints when greeting reached a process [38].

The implementation is however incomplete on this point and do not implement the in- tégralité mechanism [39]. Furthermore, its design is partly problematic. *KBlare* fails to take into account the "transitive" shared memory between two processes can be linked by a chain of shared memories and processes and thus communicate is via a shared memory area "consisting" neither has put in place.

To con? Rm usability vows continued to escape the vigilance of monitors? Ow of information, we have developed a variant of the earlier attack using projections memory? Files and shared memories . The situation is shown in? Gure 6.3 . We replace the? Le

source by a read-only memory in the projection of the sending process, the file *destination* by a projection in the read-write memory of the receiving process and? n the tube by a shared memory area in read-write by the transmitter and the receiver.

Figure 6.3 - Description of the establishment of the attack via projections files and shared memories?

We did not test directly the attack on the monitors of greeting pre- sented due to the lack of functional implementation. We performed a minimal implementation of the spread of *KBlare* described in the working heights ser [38] To play the attack. As expected, regardless of the order in which are formed the projections and the establishment of the shared memory, the greeting of? Le

source the? le *destination* is possible. In contrast, hue spread is not done properly if the projection of the? Le *source* in drive process is done last. In this case, during the screening, the color is properly propagated to drive process and to the shared memory attached to it, but does not reach the? Le *destination* projected in the writer processes attached to shared memory. With a chain of projections su? Ciently long, so it is possible to fool the mechanism.

6.2 hue propagation algorithm

We offer a formal nouveaumodèle of greeting information between containers and a formal description shades of propagation mechanisms by not considering that the vows are atomic events in the life of the system but to the operations successively *activated, executed* and *disabled.*

Tags 6.2.1, greeting and executions

We notice *C* all of a system information containers, past, present or future. In our model, all containers exist permanently. He knew? T

consider the containers no longer exists or not as containers efore Pu be the source or destination of any greeting. The information container store or convey information, from the user, other machinery, etc. The files, the memory space of the process,? The messages are containers.

The monitor greeting encodes information flow? In the system and the impact they have on the containers by attaching to each hue. In our generic model, we consider that a color is a set of *tags*. We consider, without loss of generality that all container $c \in C$ is initially carrying a single note tag t_c. This identification tag? E information originating c. We notice $T =]_{c \in C} t_c$ all tags (] denoting the disjoint union). During the life of the system, containers are created and destroyed and greeting place them. The color spread is to modi? Er all tags of the destination of a process? Ow has? N to record the fact that he could now include information from the source.

De? Nition 17 (Designed configuration, Hue). A con? Guration θ**:** $C \to \wp(T)$ **associating with each container a set of tags.** $\theta(c) = \{t_{c_1}, \ldots, t_{c_{n0}}\}$ **is the hue** c **and intuitively indicates** c **was the destination? ows originating** $c_1 \ldots, c_{not}$.

Overall Note all designed configurations possible Θ. We distinguish a particular configuration?: θ_{init}, **which** is designed initial configuration of the system as
$\forall c \in C\ \theta_{init}(c) = \{t_c\}$. **Intuitively, designed configuration represents an abstraction of the system status. This** condition develops when an event of vows takes place.

In Coq, these sets are de ned as follows?:

```
1 Variable tag: Set .
  Variable container: Set .
  Variable sourceTag: Container → Tags.
  inductive initConfiguration: Configuration: =
5 | init c:
    initConfiguration c (sourceTag c).
```

We consider three types of events related to the stream: activation, deactivation and carry-execution. Naturally, a greeting can not be executed or disabled once enabled, and can not be executed once disabled. In addition, each greeting is unique, you can not reactivate a greeting disabled. Like many greeting between two same containers can take place during the life of the system, identi? E each greeting uniquely by an element of the set F identi? ers of greeting.

De? Nition 18 (Events). Let $c_1\ c_2 \in C$ **and** $f \in F$. **It disappointed nes a relationship** $c_1 \to_f c_2$ **describing a greeting called** f **of** c_1 **at** c_2. **Event** $e \in E$ **is either a pair (** $f(v_1\ c_2)$**) with** $c_1 \xrightarrow{enable}_f c_2$ **or** $c_1 \xrightarrow{disable}_f c_2$ **is a pair (** $f(v_1\ c_2)$**) with** $c_1 \xrightarrow{exec}_f c_2$. **The first set is called** O **and the second** X. **These relationships intuitively have the following** meanings:

$c_1 \xrightarrow{enable}_f c_2$ **signi? e that the greeting identi? ed by** f **source** c_1 **and destination** c_2 **is activated ;**

$c_1 \xrightarrow{exec}_f c_2$ **signi? e that the greeting identi? ed by** f **source** c_1 **and destination** c_2 **is executed;**

$c_1 \xrightarrow{disable}_f c_2$ **signi? e that the greeting identi? ed by** f **source** c_1 **and destination** c_2 is disabled.

O contains events of activation and deactivation of greeting while *X*
contains the running events.

The evolution of the system caused by the flow? Is represented by a *execution.* We notice *E +* all non-empty event sequences of *E* and consider
E ⊂ *E +* a strict subset of those sequences, which we call *executions.* We adopt the following notations:

- *e [i]* is the *i*- th event execution *e* ∈ E;
- lg (*e)* is the length of *e,* that is to say the number of events which constitute them;
- *in]* is the pre? xed (*e* [1], . . . , *in])* length *not* of *e.*

In Coq, identi? Ers of vows constitute a set without any particular hypothesis. Events are disappointed ned as an algebraic type and executions are lists (sequences) events. To facilitate disappointed definitions, it allows empty executions in the de? Nition.

```
Variable flow: Set .

inductive Event: Set : =
| enable: container → flow → container → Event
| disable: container → flow → container → Event
| exec: container → flow → container → Event

Definition Execution: Set : = List Event.
```

The subset E of all possible sequences of events is characterized by two causal conditions, consistent with the de? nition of events given above. The activation of a greeting before his execution, above its deactivation, throughout execution.

$$\forall\, l\,[i] = c_1 \xrightarrow{disable} f\, c_2 \vee e\,[i] = c_1 \xrightarrow{exec} f\, c_2 \Rightarrow \left(\exists\, j < l\,[j] = c_1 \xrightarrow{enable} f\, c_2 \wedge (\forall\, kj < k < i \Rightarrow (e\,[k]\, 6 = c_1 \xrightarrow{disable} f\, c_2) \right) \tag{6.1}$$

In Coq, the causal property is expressed demanière inductive. The empty implementation is causal, and any execution composed of a causal sub-execution to which is added an event following a few assumptions (deactivation can be added that if the activation is in the sub-execution, for example).

```
inductive causality: Execution → Prop : =
| causality_nil: causality []
| causality_enable c1 f c2 e
    ( Hind: causality e)
    ( Hnoenable: ¬ In (enable c1 f c2) e): causality (enable
    c1 f c2 :: e)
| causality_exec c1 f c2 e
    ( Hind: causality e)
    ( Henable: In (enable c1 f c2) e)
    ( Hnodisable: ¬ In (disable c1 f c2) e): causality (c1 f c2 ::
    exec e)
| causality_disable c1 f c2 e
    ( Hind: causality e)
```

```
    ( Henable: In (enable c1 f c2) e)
    ( Hnodisable: ¬ In (disable c1 f c2) e): causality (disable
    c1 f c2 :: e).
```

We assume that all the events of execution can not be observed by a monitor? Ow of information. For example, using a monitor LSM can view and change? er system status when passing in a hook. In particular, we ask that O is the set of observable events while events X are hidden, that is to say they are not detectable by the monitor of greeting. This models the fact that the monitor is implemented with LSM and therefore can not observe the entire system activity. In fact, he can be aware of the system status and e? Ectuer the spread of hues when executing a system call reaches a hook LSM . The actual operation of the system call, the vows, is performed between the first hook and a second event allowing the monitor to act as a hook placed to? N of the system call for greeting discrete or in the call system? n of greeting for greeting continuous.

It disappointed naturally nit observability of events and happenings in Coq. Function flatten_list used to meet the Coq type system because causality applies to lists of events and not of observable events listings.

```
inductive observable: Event → Prop : =
| enable_is_obs c1 f c2:
    observable (enable c1 f c2)
| disable_is_obs c1 f c2:
    observable (disable c1 f c2).
inductive hidden: Event → Prop : =
| exec_is_hidden c1 f c2:
    hidden (exec c1 f c2).

Definition ObsExecution: =
  { e: list {ev: Event | observable ev} |
    causality (flatten_list e)}.
Definition HidExecution: {list = ev: Event | ev hidden}.
```

We notice O all observable executions, that is to say containing only events in O, and X all hidden executions, containing only events X. Observable executions represent that a flow of information monitor is able to see the hidden while executions are those that represent the evolution e? Ective information containers. Is $e \in$ E, we write

$e_O \in$ O (respectively $e_X \in$ X) the observable performance (the performance hidden respectively) obtained by removing the hidden events (observable events respectively) of e. Both projections of execution e are linked by causal conditions set out in equation (6.1). It disappointed nes therefore a compatibility relationship between an observable performance and a hidden execution.

De? Nition 19 (Compatibility). An observable execution ω is compatible with a hidden execution x if, and only if, they are projecting an even execution E. formally,

$\forall x \in$ X $\forall \omega \in$ O, we write $\omega \; x$ iff $\exists e \in$ E ($\omega = e_O \wedge x = e_X$).

In Coq, we de? Ne the inductively compatibility. In a way, the de? Nition gives a relationship between performance and both observable and hidden under-executions composing, depending running with one less item. We then prove a theorem showing that this de? Nition is equivalent to the de? Nition 19 If we compatibility relationship between an observable performance, a hidden implementation and execution, so that execution respects causality and is composed of events hidden and observable executions.

```
inductive Compatible: ObsExecution → HidExecution → Execution → Prop : =
| compatible_empty:
    Compatible (exist _ [] causality_nil) [] []
| compatible_add_exec
    oxe c1 f c2
    ( Hcompatible: oxe compliant)
    ( Henabled: In (make_enable c1 f c2) ( 'o))
    ( Hnot_disabled: ¬ In (make_disable c1 f c2) (o): o Compatible (make_exec c1
    f c2 :: x) (c1 f c2 :: exec e)
| compatible_add_enable
    oxe c1 f c2
    ( Hcompatible: oxe compliant)
    ( Hno_enabled: ¬ In (make_enable c1 f c2) (o)): Compatible (make_enable_causality c1 f c2 o Hno_enabled) x (enable
    c1 f c2 :: e)
| compatible_add_disable
    oxe c1 f c2
    ( Hcompatible: oxe compliant)
    ( Henabled: In (make_enable c1 f c2) ( 'o))
    ( Hnot_disabled: ¬ In (make_disable c1 f c2) ( 'o)):
    Compatible (make_disable_causality c1 f c2 o Henabled Hnot_disabled) x (
        disable c1 f c2 :: e).

Program Theorem compatibility_implies_interleaving_respecting_causality:
 ∀ ( o: ObsExecution) (X: HidExecution) (e: Execution) compatible oxe → ( e causality
      ∧ ∀ ev (In ev (flatten_list (o)) ∨ In ev (flatten_list x)) ↔ In ev e).
```

Example 1. Consider the first attack presented in section 6.1.1 and illus- tree in? gure 6.4 . In the rest of this article, we shortened the name containers as follows: *src* is here *source, himself* the process *reader* of the *source (* and writer of the tube) *p* the *tube, r* the process *writer* of *destination (* and tube reader) and *d* the

destination. The column *x* represents the hidden execution while column *ω* corresponds to the observable performance. Both versions are compatible for running the column *e* interlaces *x* and *ω* so that the causal conditions (6.1) Are met.

6.2.2 Interpretation of executions in terms of greeting infor- mation

It is said of a propagation mechanism of tints it is *correct* if it takes into account all the vows that took place, that is to say if it does not there a hidden execution consistent with the observation observed by the monitor, such as a

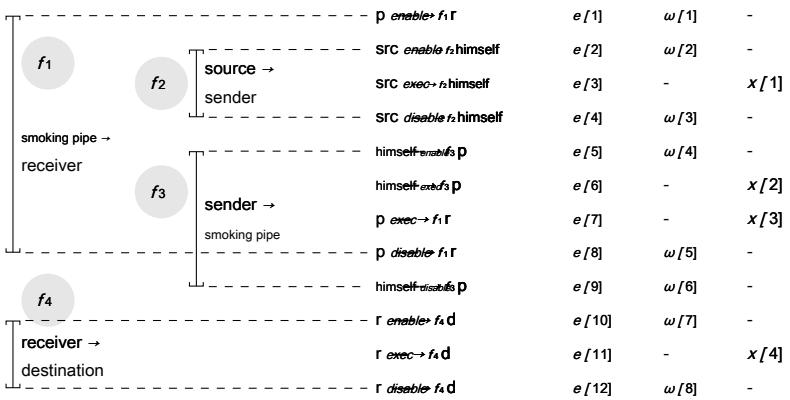


Figure 6.4 - Executions observable and hidden from the example of attack

ow executed is not re? been in spreading shade. Naturally, an observable exécu- tion can be compatible with several hidden killings, since many vows are activated simultaneously. A hidden execution is actually a total order on the flow occurring in the system while an observable performance is a partial order on the stream???? ux appear some competition in the eyes of the monitor ow, they can not not ordered temporally. All hidden executions consistent with observable execution is exactly the set of total orders consistent with this partial order. Therefore, a monitor? Ow of information can not claim an exact color spread in the general case.

6.2.3 Propagation ideal

It disappointed defines a transition relation ↪ - → ⊆ Θ X × × Θ between designed configurations describing how a monitor? ow of information that could observe the executions of ux themselves rather than their activation and deactivation e? ectuerait spreading shade.

$$\theta \xrightarrow{exec} \theta[c_2 \leftarrow \theta(c_2) \cup \theta(c_1)]$$

The spread of hues simply to add to the label of the destination all tags from the source. Is $x \in X$ we notice $\theta_0 \xrightarrow{x[:n]} \theta_{not}$ when it was $\theta_0 \xrightarrow{x[1]} \theta_1 \xrightarrow{x[2]} \cdots \theta_{not-1} \xrightarrow{x[n]} \theta_{not}$. Using the example 1 page 137 , table 6.1a details the calculation of an ideal propagation? ectuée by $x \rightsquigarrow$.

In Coq, we de? Ne a hidden execution as a relationship between designed gu- rations. Two con? Gurations are related, if the designed configuration arrival is identical to the designed starting configuration except in terms of the destination of the greeting, which includes all the tags of the source of it.

```
1 inductive ConcreteExecution1: Configuration → {e: Event | hidden e} →
        Configuration → Prop : =
```

Table 6.1 - Interpretation of executions.

In the following tables, we note: t_{src} = F t_{se} =, t_p =, t_r =, t_d = NOT

(A) Calculation of θ_{init} $\xrightarrow{x[:n]}$ θ (ideal propagation)

not	x [n]	θ (src)	θ (is)	θ (p)	θ (r)	θ (d)
1	*src* exec ---→ f_1 *himself* F		, F?		}	NOT
2	*himself* exec f_2 *p* F		, F?	, , F}		NOT
3	*p* exec → f_3 *r* F		, F?	, , F} ?	, , FN	
4	*r* exec → f_4 *d* F		, F?	, , F} ?	, , FN} ?	, , F

(B) Calculation of θ_{init} $\xrightarrow{\omega[:n]}$ θ (Monitors greeting based on LSM)

not	ω [n]	θ (src)	θ (is)	θ (p)	θ (r)	θ (d)
1	*p* enable→ f_1 *r*	F			} ?	NOT
2	*src* enable ---→ f_2 *himself* F		, F?		} ?	NOT
4	*himself* enable f_3 *p*	F	, F?	, , F} ?		NOT
7	*r* enable→ f_4 *d*	F	, F?	, , F} ?		NOT, }, ?

```
| exec_conc (theta1: Configuration) c1 f c2 (theta2: Configuration)
  ( Htheta: ∀ ct theta2 ct ↔ ((c = c2) ∧ theta1 c1 t) ∨ theta1 ct): ConcreteExecution1 theta1
  (make_exec c1 f c2) theta2.

inductive ConcreteExecution: Configuration → HidExecution → Configuration
        →  Prop : =
| exec_conc_refl (theta: Configuration): ConcreteExecution
  theta [] theta
| exec_conc_step (theta1 theta theta2: Configuration) (x: HidExecution)
    ( ev: Event) (Hhid: hidden ev)
    ( Hstep: ConcreteExecution1 theta (ev _ exist Hhid) theta2)
    ( Hind: ConcreteExecution theta1 x theta): ConcreteExecution theta1 (x make_hidden_exec
  Hhid ev) theta2.
```

6.2.4 Propagation of shades e? Ectuée by implé- mented monitors with LSM

We now describe how the monitors? Ow of information designed based hooks LSM , as *Laminar, KBlare* or *Weir* spread the tints. The main feature of these monitors is that they are based only on hooks

LSM placed before the flow?. Therefore, in our model, this means that they only account activation events and consider them as equivalent to the execution of greeting events. Formally, the hue of propagation of computation performed by these monitors can be described by a relation

- ⇁ ⊆ Θ O × × Θ ? De ned by:

$$\theta\ c_1 \xrightarrow{\text{enable}} c_2 \quad \theta\,[c_2 \leftarrow \theta\,(c_2) \cup \theta\,(c_1)] \qquad\qquad \theta\ c_1 \xrightarrow{\text{disable}} c_2 \quad \theta$$

When an activation event occurs, the greeting instructor hastens to e? Ectuer the spread of hues. However, it ignores the disabling events. Considering again the example of the situation 1 , table 6.1b describes the computation performed by ω

- ⇁. This spread is not the correction of property: while ω ` x the flow of indirect? *source* towards *destination* is missed by ω

- ⇁ (t_{src} = F ∈ θ (d) = { t_d =

NOT, t = r} t_p = , t_{sc} = , t_{src} = F} in computing x ⇢ , but this is not the case for ω - ⇁).

There are actually two tracking models ux at the exploi- tation system: that of *shades? oating* used by *KBlare* and *Weir* and that of *explicit elevations* used by *Laminar.* In the model of labels? Oats, the first to have been formalized and used, for example in *IX* [56], When a greeting is detected, the color of the destination is automatically increased with the hue of the source. However, in an explicit elevation model, the process causing the stream? Must modi? Er explicitly the hue of the destination before e? Ectuer the flow?. This model has been formalized by Zeldovich and his collaborators, designers of *HiSTAR* [109]. The model proposed here described directly ux? Tracking model shade? Oating but less intuitively, it also correctly describes the explicit elevation model labels. In e? And, in both cases, the competitive environment which we illustrate in Section 6.1.1 exists and has the same e? ect. In *KBlare* and *Weir* the flux? of the tube receiving process takes place without the receiver's color is updated correctly because the ux? are observed in the reverse order of their execution. If the greeting is illegal according to some current security policy in the system, then the violation of this policy is not detected and no alerts are raised. In

Laminar, even if the greeting of the tube to the receiver process is illegal *when it is e? ectué,* which means? e that the receiver does not allow tint greeting normally, it is not yet *when it is observed,* and it can be triggered without raising hue and receiver without raising alarm. In fact, in our model, the colors represent the monitor's knowledge of greeting information about greeting past system, and ignores their precise semantics in terms of policy? Ux of information.

6.2.5 Smaller correct overapproximation tints propa- ger

A? N de? Ne a proper hue spread, we de? Ne first all of all vows, direct and indirect, which may be caused by an observable performance. We recall that all these vows correspond to all direct and indirect flow caused by any hidden executions consistent with the observed performance.

Given an observable performance ω, one dice? nes $Enabled_\omega \subseteq C \times C$ as in- seems the vows have been activated during the execution ω but not deactivated at? n of it ω.

$$(c_1\ c_2) \in Enabled_\omega \Leftrightarrow \exists\, i\ \omega\,[i] = c_1 \xrightarrow{\text{enable}} c_2 \wedge \forall\, j > i\ \omega\,[j]\ 6 = c_1 \xrightarrow{\text{disable}} c_2$$

Enabled $^{\star}_{\omega}$ closing is re? exive and transitive the relationship *Enabled* $_{\omega}$. All ow? That may be caused by ω writes *flows* $_{\omega} \subseteq C \times C$ and is calculated as follows. 1

$$\textit{flows}_{\omega} = \begin{cases} \textit{Enabled}^{\star}_{\omega} & \text{if lg } (\omega) = 1 \\ \textit{flows}_{\omega[k]} \; \textit{Enabled}^{\star}_{\omega} & \text{if lg } (\omega) = k + 1 \end{cases}$$

In Coq, we de? Ne the sets of vows activated as above and in- seems the vows went inductively, beginning with the execution comprising a single event.

```
1 inductive Enabled: ObsExecution → container → container → Prop : =
  | enabled_if_not_disabled_yet o c1 f c2
    ( Henabled: In (make_enable c1 f c2) ( 'o))
    ( Hnot_disabled: ¬ In (make_disable c1 f c2) ( 'o)):
5   O Enabled c1 c2.

  inductive flows: ObsExecution → container → container → Prop : =
  | flows_1 (c1 ' c2 ' c1 c2: Container) (f: Flow)
      ( Henabled: (Clos_refl_trans _ (Enabled (c1 make_singleton_causality ' f c2 '))
        c1 c2):
10    Flows (c1 make_singleton_causality ' f c2 ') c1 c2
  | flows_ind_enabled (o: ObsExecution) (c0 c1 c2 c1 ' c2 ': Container) (f: Flow)
    ( Hind: Flows o c0 c1)
    ( Hnot_enabled: ¬ In (c1 make_enable ' f c2 ') ( O))
    ( Hlast: (Clos_refl_trans _ (Enabled (c1 make_enable_causality ' f c2 ' o
        Hnot_enabled)) c1 c2)):
15    Flows (c1 make_enable_causality ' f c2 ' o Hnot_enabled) c0 c2
  | flows_ind_disabled (o: ObsExecution) (c0 c1 c2 c1 ' c2 ': Container) (f: Flow)
    ( Hind: Flows o c0 c1)
    ( Henabled: In (c1 make_enable ' f c2 ') ( O))
    ( Hnot_disabled: ¬ In (c1 make_disable ' f c2 ') ( O))
20    ( Hlast: (Clos_refl_trans _ (Enabled (c1 make_disable_causality ' f c2 ' o
        Henabled Hnot_disabled)) c1 c2)): Flows (c1 make_disable_causality ' f c2 ' o Henabled Hnot_disabled) c0
      c2.
```

For instance, if the greeting (*A, B)* occurred during the observable current execution, and the vows (*B, C)* is activated, then the composition (*A, C)* is a new greeting (as (*B, C)* naturally). This would not be the case if (*B, C)* predated (*A, B)*. Still considering the example 1 , table 6.2a illustrates the calculation of *flows* $_{\omega}$. *flows* $_{\omega}$ is not necessarily a transitive relation, because the order of greeting account.

We prove in the proposal 6 below that this calculation of *flows* guarantee the correction of the propagation of colors associated, illustrated in Table 6.2b . Proposal 7 guarantees meanwhile it is impossible to calculate a smaller surappoximation hues ensuring the correction in our model. In other words, for every greeting appearing in *flows* $_{\omega}$ there is a hidden execution compatible with ω causing this greeting. To the extent that these proposals have been demonstrated with Rooster, we give here only intuitions of evidence and we refer to Annex C for details.

---

1. Consider two relationships $R_1 \subseteq E \times F$ and $R_2 \subseteq F \times G$, the relationship $R_1 R_2 \subseteq E \times G$ is de? ned by $(x, y) \in R_1 R_2$ if, and only if, there is $z \in F$ such as $(x, z) \in R_1$ and $(z, y) \in R_2$.

Proposal 6 (Correction). *All vows engendered by an observable performance ω belong to flows ω.*

$$\forall e \in \mathrm{E}\ \forall \theta \in \Theta\ \theta_{init} \quad \xrightarrow{ex}{}^{*} \theta \Rightarrow \forall c \in C\ \theta(c) \subseteq \bigcup_{(c,c) \in flows_e o} \theta_{init}(c')$$

Primer evidence. By induction on lg (*e).* He knew? T show that if a hidden execution exists, then, for the causal conditions (6.1), There is necessarily a sequence of observable events that enabled the flow? The hidden performance. By construction, these vows are therefore in *flows ω.*

□

Correction of property is written as follows in Coq. If there is a designed concrete-compatible configuration with an observable performance data and providing a tag *t* a container *c,* then there is a greeting from a container *c* ' who wore this tag *t* in designed to initial setup *c.* In other words, all concrete designs are considered in the relationship *Flows.*

```
Program Theorem soundness:
  ∀ oxe, e 6 = [] → oxe compliant → ∀ theta ConcreteExecution initConfiguration x theta →
  ∀ ( c: Container) (t: Tag), theta ct →

  ( exists c ' flows oc ' c ∧ initConfiguration c ' t).
```

Proposal 7 (Smaller overapproximation / Completeness). *All? Ow flows ω are generated by at least one hidden performance compatible with ω.*

$$\forall \omega \in \mathrm{O}\ \forall c, c' \in C \quad (\ c, c') \in flows_\omega \Rightarrow \exists x \in \mathrm{X}\ (\ `\omega\ x \wedge \forall \theta \in \Theta\ \theta_{init} \xrightarrow{x}{}^{*} \theta \Rightarrow \theta_{init}(c) \subseteq \theta(c')\ )$$

Primer evidence. By induction on lg (*ω).* Suppose we have (*c, c* ') ∈ *flows ω [: n]* a greeting compound (*c = c*1 *c*2) (*c*2 *c*3), . . . (*cm* - 1 *cm = c* '). Then, by de? Nition, there *i ≤ m* such as (*c*1 *ci*) ∈ *flows ω [: n* - 1] and (*ci cm*) ∈ *Enabled* ∗ *ω [: n].* By the induction hypothesis there *ω x '[: n* - 1] propagating the tag *c*1 at *ci.* By concatenating *x* with the sequences of ux? (*ci ci* + 1), . . . (*cm* - 1 *cm*) (that are enabled but not disabled in *ω [: n])* in that order, a hidden performance is obtained

x · *ω [: n]* causing propagation tag *c = c*1 at *cm = c* ' via *ci.* □

The following proposal is writing in the Coq theorem stating that there are no more small relationship *flows* with all good wishes possible. That is to say that if a? Ow *c*1 towards *c*2 exists in *flows,* there is necessarily an observable performance supports transferring the tags *c*1 towards *c*2.

```
Program Theorem completeness:
  ∀ ( o: ObsExecution) (C1 c2: Container) Flows o c1 c2
  → ∃ ( x: HidExecution) (E: Execution)

  ( oxe compliant ∧
   ( forall theta ConcreteExecution initConfiguration x theta →
    ( forall t initConfiguration c1 t → theta c2 t))).
```

Table 6.2 - Application example of the further spread of hues

(at) *Enabled* ω [: n], *Enabled* * ω [: n] and *flows* ω [: n]

not	*Enabled* ω [: n]	*Enabled* * ω [: n]	*flows* ω [: n]
0		(*src, src) (is, was), (p, p),* (*r, r), (d, d)*	(*src, src) (is, was), (p, p), (r, r),* (*d, d)*
1	(*p, r)*	(*src, src) (is, was), (p, p),* (*r, r), (d, d), (p, r)*	(*src, src) (is, was), (p, p), (r, r),* (*d, d), (p, r)*
2	(*p, r),* (*src, se)*	(*src, src) (is, was), (p, p),* (*r, r), (d, d), (p, r), (src, se)*	(*src, src) (is, was), (p, p), (r, r),* (*d, d), (p, r), (src, se)*
3	(*Sep),* (*src, se),* (*p, src)*	(*src, src) (is, was), (p, p),* (*r, r), (d, d), (if, src) (where, p),* (*src, p), (src, se), (p, se),* (*p, src)*	(*src, src) (is, was), (p, p), (r, r),* (*d, d), (p, src), (p, se), (src, se),* (*src, p), (if, src) (where, p)*
456		(*src, src) (is, was), (p, p),* (*r, r), (d, d)*	(*src, src) (is, was), (p, p), (r, r),* (*d, d), (p, src), (p, se), (src, se),* (*src, p), (if, src) (where, p)*
7	(*r, src)*	(*src, src) (is, was), (p, p),* (*r, r), (d, d), (r, src)*	(*src, src) (is, was), (p, p), (r, r),* (*d, d), (p, src), (p, se), (src, se),* (*src, p), (if, src) (where, p), (r, src)*
8	(*r, d)*	(*src, src) (is, was), (p, p),* (*r, r), (d, d), (r, d)*	(*src, src) (is, was), (p, p), (r, r),* (*d, d), (p, r), (p, d), (src, se),* (*src, p), (src, r), (src, d), (where, p),* (*to r) (where, d)*

(B) Calculation of $\bigcup_{(c1,\,c2)\,\in\,\mathit{flows}\,\omega\,[:n]} \theta_{\mathit{init}}(c1)$ (Overapproximation of Rfblare)

not	*ω [n]*	*θ (src)*	*θ (is)*	*θ (p)*	*θ (r)*	*θ (d)*
1	*p* enable→ f_1 *r* F				} ?	NOT
2	*src* enable→ f_2 *himself* F		, F?		} ?	NOT
3	*src* disable----- → f_2 *himself* F		, F?		} ?	NOT
4	*himself* enable f_3 *p* F		, F?	, , F} ?	, , FN	
5	*p* disable→ f_1 *r* F		, F?	, , F} ?	, , FN	
6	*himself* enable f_3 *p* F		, F?	, , F} ?	, , FN	
7	*r* enable→ f_4 *d* F		, F?	, , F} ?	, , FN} ?	, , F
8	*r* disable→ f_4 *d* F		, F?	, , F} ?	, , FN} ?	, , F

6.3 Implementation and Experiments

6.4 Design

We implemented our hue propagation algorithm as a security module LSM called *Rfblare (* for *race-free Blare, Blare* without race condition) by taking the basic code *KBlare.* We used version

4.7 kernel o? Heaven, the latest available at the time of project start. We only implemented the spread of colors, and have not contributed to the im- plémentation security policies, or the spread of hues on the network implemented by *KBlare.* We do not discuss these points here. The system calls monitored by *Rfblare* are listed in the table 6.3 . It can be seen at first that *Rfblare* aims to cover a large number of open channels, some well-known as read, write and their derivatives, other speci? c to Linux as

process_vm_readv, although we can not claim to be exhaustive.

According to the formal description of the algorithm, we use a hook LSM as an activation event ux? and one as désacti- vation event. The projection of our model of the kernel code is not immediate and the knowledge gained during the work presented in Chapter 5 we served here. It is interesting to note that disabling event is not always necessary. In e? And some system calls can not get in race condition with others as the container on which they act is also locked, or is created by the system call itself. This is the case of the sys- tem call exec for example. This call replaces the memory of the calling process to run a new program. When the brackets LSM bprm_set_creds

and bprm_committing_creds are called the? le is locked and can not be the destination of greeting. The memory of the new process on it is about to be completely replaced and the new memory is not yet installed. It therefore can not be the source of greeting. In the case of fork, the situation is similar. The hook mm_dup_security is not set to control the creation of the new process but only to duplicate the security structure associated with memory. This duplication is made at a point where the newly created memory can not be neither the source nor the destination of a stream; therefore, there is no possible competition conditions. In? N, mq_timedsend and msgsnd are also special cases. When a message is sent to a? The message, it is first copied to a kernel memory buffer. The hook LSM is called at that time, before inserting e? ective in the message? the. However, as the message is already copied, it can not be the target of greeting. This is deliberate, and prevents the sending process manipulates the message after veri? Cation. The kernel already avoids the condition of competition in the information container (the message) in this case, there is no need for *Rblare* adding its own mechanism, which would be redundant.

Most brackets corresponding to activation events already present, as we have veri? Ed in Chapter 5 . brackets LSM corresponding to off events due to be added however. We simply two hooks di? Erent, because according to our model, the spread of greeting is actually e? Ectuée to activation of greeting, and deactivation of actually serves to maintain the list of vows enabled. The hooks are added

syscall_before_return for greeting and discrete ptrace_unlink firstly to

process_vm_readv and process_vm_writev (who are greeting discrete) and secondly to ptrace (ux continuous). ptrace is a call particular system used principally for debugging and diagnostics. It allows a process to attach to another and to monitor its implementation. The controller can process particularly manipulate registers and memory attached process. In reality, memory manipulation tions demanding another call ptrace even once an attachment? ectué, so one could consider each of these vows as greeting discreet. However, ptrace opens many channels open and hidden communication between the two processes, which depend more than architecture. It is more convenient to consider ptrace as a greeting continuously. In terms of greeting dis- decrees, we used the new hook syscall_before_return, placed just before the return of system calls concerned. This hook is used only to withdraw from the list of vows activated the vows being whatsoever. The only exception is

process_vm_readv and process_vm_writev. These two system calls already had the hook ptrace_access_check also used by ptrace. To ensure consistency in our list of vows activated, and for simplicity we have used in this case the same hook? N for ptrace. The case of shmat, mmap and mprotect is specific. Unlike other greeting where *Rfblare* maintains itself information activation and deactivation, the greeting caused by? le memory mapped is managed by querying the kernel itself on active projections. ? Specifically, for each file, the kernel maintains a list of process memory in which it is intended; and for every memory we can also know the files? that are projected into it. The kernel maintains this knowledge for its own use, in particular the proper release of resources. So we have always an accurate graph and day, never out of sync with reality, regarding the flow due to the? Le projected. We use this graph to spread the tints appropriate manner. We still use the hooks LSM activation of the greeting has? n propagate the colors at the appearance of a projection. The projection is unidi--directional (the? Le to the process) if it is in one and two-way play if it is read-write. As mprotect can change the permissions of a read-only projection read-write, it is important to take into account.

The heart of *Rfblare* is a simple table of? ux activated doubly indexed by the source of? ux (which therefore forms a graph) and the process that triggered the greeting (essentially? n to allow deactivation of? ux and the release of resources). With each new appearance of a greeting, a graph traversal of vows is made has enabled? N to spread the shades at all attainable containers. Besides the graph maintained by *Rfblare,* we also use the graph of? le projected as described above. The path width converges because the calculated function is monotonous. You can stop the course of a path graph from the meeting of a node already have tags from the source of the greeting.

6.5 Tests

We tested the e attacks? Ectuées on *KBlare* and presented in section 6.1.1 a? n to validate the proper operation of the spread of color. In the case of the first attack exploiting the competitive environment on the tubes, the sequence of events is of course the same as that shown in? Gure 6.1 but *Rfblare* properly spread the tints. The first event is the activation of the flux tube? To the receiver. The greeting is stored as active at that time then the receiver

Table 6.3 - RSS monitored by *Rfblare* and LSM hooks used system call

	activation	Deactivation
Discrete? Ows		
read.	file_permission *at*	before_return *c*
readv.	file_permission *at*	before_return *c*
preadv.	file_permission *at*	before_return *c*
pread64.	file_permission *at*	before_return *c*
write.	file_permission *at*	before_return *c*
writev.	file_permission *at*	before_return *c*
pwritev.	file_permission *at*	before_return *c*
pwrite64.	file_permission *at*	before_return *c*
sendfile.	file_permission *at*	before_return *c*
sendfile64.	file_permission *at*	before_return *c*
recv.	socket_recvmsg *at*	before_return *c*
recvmsg.	socket_recvmsg *at*	before_return *c*
recvmmsg.	socket_recvmsg *at*	before_return *c*
recvfrom.	socket_recvmsg *at*	before_return *c*
send.	socket_sendmsg *at*	before_return *c*
sendmsg.	socket_sendmsg *at*	before_return *c*
sendmmsg.	socket_sendmsg *at*	before_return *c*
sendto.	socket_sendmsg *at*	before_return *c*
process_vm_readv.	ptrace_access_check *at*	ptrace_unlink *c*
process_vm_writev	ptrace_access_check *at*	ptrace_unlink *c*
migrate_pages. . . .	task_movememory *at*	before_return *c*
move_pages.	task_movememory *at*	before_return *c*
fork.	mm_dup_security *c*	- *d*
clone.	mm_dup_security *c*	- *d*
execve.	bprm_set_creds *at /*	- *d*
execveat.	bprm_committing_creds *at* bprm_set_creds *at /* bprm_committing_creds *at*	- *d*
msgrcv.	mq_store_msg *at*	before_return *c*
msgsnd.	msg_msg_alloc_security *at*	- *d*
mq_timedreceive. .	mq_store_msg *b*	before_return *c*
mq_timedsend.	msg_msg_alloc_security *b*	- *d*
continuous flow		
shmat.	mmap_file *at*	- *e*
mmap_pgoff.	mmap_file *at*	- *e*
mmap.	mmap_file *at*	- *e*
ptrace.	ptrace_access_check *at*	ptrace_unlink *c*
ptrace.	PTRACE_TRACEME *at*	ptrace_unlink *c*

at - Hook LSM already present. *b* - Hook LSM existing and reused for this system call.

c - Hook added by us.

d - No hook LSM necessary because the call can not compete with another on the same container.

e - No hook LSM necessary because the information greeting activated on? Ie projected is obtained by other means, by querying the core.

falls asleep before leaving the system call. Then, the transmitter activates the process? Ux of? Source file to its memory. At this point, the tag? Source file is transferred to the transmitter. Since no greeting having source for the transmitter is active, the propagation stops at this point. The ux? Is then deactivated after reading the? Le and the greeting is removed from the list of vows active. The next event is the activation of greeting writing in the tube by the issuer. At this point, the stream? Tube to the receiver is always active, and therefore the color of the issuer, containing the tag of? Le, is transmitted to the tube and the receiver. In? N, the stream? Are disabled, and the greeting of the receiver? Shit destination place. The initial tag? Source file, and that of all containers of the path of vows, is found in the color of the? Le destination.

We also reproduces the second attack using? Files memory mapped and shared memory area to replace the tube. In this attack, we can vary the order in which the projections and shared memory are implemented, always making sure to copy the contents of the source file in the shared memory and shared memory in the? Le destination. Thanks to kernel data structures allowing us to know the correspondence between? Projected files and processes memories, *Rfblare* is able in every case to spread the colors correctly.

We also conducted experiments to evaluate the overhead performance driven by employment *Rfblare,* since its shades propagation algorithm in practice requires calculating a transitive Closing time graphs. The measures that we practice are not directly comparable with other monitors vows because we do not measure the cost of veri? Cation of the legality of greeting, no reaction (stop the offending process, waiver of alerts, communications to a policy manager implemented in user space, etc.). However, the measures characterize exactly what di? Érencie *Rfblare* other monitors vows, namely the spread. We have established a test case, which is to compile the Linux 4.7 kernel, on a system where *Rfblare* is installed. We made a first compilation to see what? Les were involved in compiling the kernel image? Nal, using the symbol table and kernel debugging information. We then used that list to assign a unique tag to a variable number of these? Les. Then we raise a compilation to study the impact of the number of tags to spread the speed of compilation. The measurements were performed on a physical machine, having 16 GB RAM and eight cores of 3.2 GHz two threads each. Each measurement was replicated at least thirty times. We chose the compilation as test cases for several reasons. First, it is a reproducible test. Second, the compilation involves a lot of input-output and manipulation? Les, and the process of creation. In? No, it is possible to check? Widespread er of good Operates in *Rfblare* watching what tags are propagated in the final image and the? le intermediate and comparing this information with the symbol table and debug information. Our results are presented in the table 6.4 . It may be noted that in this test, the impact of *Rfblare* is invisible and not measurable. Older measurements showed an increased overhead of a few percent, but the latest versions of *Rfblare* will show more di? erence compared to a coreless *Rfblare.* These results indicate that in this case, the bottleneck is not the spread of shades but probably

the input-output or synchronization between the processes involved in compiling. Even if the experiment we conducted does not test a worst case (but realistic case anyway), we con? Rmation that the spread of colors that we offer does not appear at first prohibitively expensive in terms of execution time.

6.6 Conclusion

The work described in this chapter were presented at the conference MPAC, 2017 (*Software Engineering and Formal Methods) [* 34]. The article [34] Was awarded the *Best Paper Award*. We proposed here is essentially a new way to e? Ectuer the spread of hues in a Linux security module based on the *framework* LSM . Unlike previous approaches in the field, we are able to process:

- ? Ows caused by process e ectuant competing system and calls on the same information containers;
- the information consisting of the? le memory mapped channel, including memory shared between processes.

In reality, the first point is a necessary condition for the second. In e? And the? Ow in memory, and vows continuous generally are an example of greeting that can compete with many others. If the pairs of process A and B on the one hand, and B and C on the other, respectively share a memory area, then it is necessary that the monitor of greeting information acts as if A and C share a memory area also. In reality, this memory really does not exist and is only a composition of genuine shared memory areas.

We achieve these goals by not considering the greeting as atomic events but activation sequences, possible executions and off, with the peculiarity that only activations and deactivations are observable by the monitors of greeting information implemented with LSM Because they correspond to hooks LSM . In the particular case of the greeting caused by projections memory? Les, which includes shared memory, it does not seek to detect the activation and deactivation but directly interrogates the kernel, when planning the spread, to know the greeting enabled. Our approach requires changing many aspects of the core since we need to views of the? N? Ows. In our prototype, we did adding hooks

framework LSM . Should this be undesirable, it would be possible instead to implement a mechanism to observe all the system calls back and consider these as points of disabling greeting discreet. However, this solution would not be easy to implement in a portable way, since the return of the user space kernel space is implemented di? Erently according architectures.

This contribution also allowed to exhibit some limitations of the contribution described in the previous chapter. For example, in the family system calls splice, in the case of a greeting from a? le to a tube, a hook LSM is crossed but it is the hook file_permission. The greeting that is observed by the monitor is the same as in the case of reading the? Le, that is to say a stream of? Le to the memory of the current process, not the? Le to the tube. It is not known of the monitor. To e? Ectuer propagation correct colors in this case, it is necessary in the core which is implemented to change the hook Rfblare

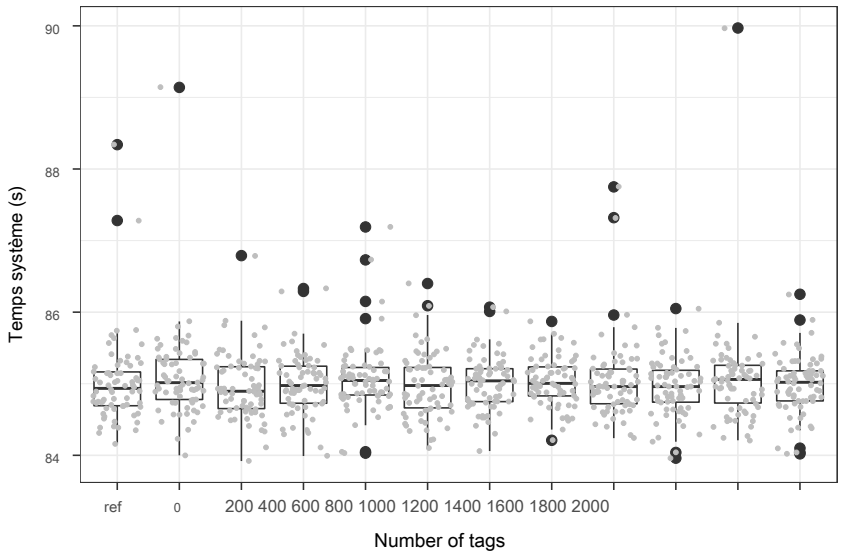
Table 6.4 - Result of micro- *benchmark* compilation Number

User tags	Time (s)	System Time (s)	Elapsed Time (s)
ref.	1134 ± 1.06	85.02 ± 0145	211.4 ± 0.42
0	1136 ± 1.03	85.10 ± 0148	211.9 ± 0.57
200	1136 ± 1.11	84.96 ± 0108	212.0 ± 0.66
400	1136 ± 1.03	85.00 ± 0101	212.2 ± 0.57
600	1136 ± 1.08	85.07 ± 0112	211.5 ± 0.37
800	1135 ± 1.02	84.98 ± 0107	211.7 ± 0.48
1000	1136 ± 1.09	85.00 ± 0087	212.1 ± 0.57
1200	1136 ± 0.85	85.03 ± 0081	211.6 ± 0.52
1400	1137 ± 0.96	85.04 ± 0131	212.2 ± 0.54
1600	1135 ± 1.10	84.96 ± 0096	212.1 ± 0.55
1800	1136 ± 0.84	85.06 ± 0166	212.0 ± 0.41
2000	1136 ± 1.04	84.99 ± 0095	211.9 ± 0.48

The times are given as an average over thirty measures at least with their interval con? Dence 95%.

The first column identi? Es the test cases. The reference is a system identical to that of the other cases, except that *Rfblare* is not used. The remaining cases is identi? Ed by the nmbre of source les bearing a tag at the beginning of the compilation.
"User Time" column shows the cumulative time spent by the compiler outside the nucleus, in user space.

"System Time" column gives the inverse time spent in the kernel to execute system calls, and therefore among others, e? Ectuer the spread of hues. The "Elapsed Time" column shows the total duration of the compilation. This value is di? Erent from the sum of the two previous values because it does not stack the time spent by di? Erent processes and compilation of threads running in parallel.



The points in light gray are the individual measures, àmoustache boxes give quartiles measures, the black points are outliers.

another, taking as parameter the? le and the tube. This opens the way for future work on a formal semantics of system calls in terms of greeting information and static analysis of the correspondence between the hooks LSM

crossed into system calls and these vows. This reminds work e? Ectué Jaeger, Edwards and Zhang in "Consistency analysis of authorization hook investment in the Linux security framework modules" [42] Except that this new analysis would control? Ow of information and not the access control.

chapter 7

Conclusion

We discussed in the context of this thesis several types of problems. A software engineering problem first. The Linux kernel is rich, at the time of writing, twenty-five years of eventful history ever since Linus Torvalds advertised on Usenet his newly created kernel. Several thousand? Les, and several million lines of code, accumulated to form the nucleus as it can be used today on many architec- tures and types of equipment di? Erent from the Android phones to supercomputers. Our work on the analysis of properties of the kernel code, the first work of this thesis has focused on understanding the code. The expertise gained during this phase is re-usable. We implemented the Kayrebt project GCC and a graphical interface for viewing the collection? graphs ot control extracts a code base.

The second problem we are attacked is a semantically tick problem. It is on the way to analyze the system call code producing greeting information to con? Er the right position hooks LSM . For this we have disappointed nor a model code, based on graphs of? Ow control as extracted by the code Kayrebt :: Extractor. We have provided these graphs a formal semantics, we have proven correct vis-à-vis common and simple assumptions about the concrete semantics of the code. The analysis that we have developed has allowed us to determine precisely how the *framework* LSM tracks the? ow of information. We concluded that this *framework* e is? adapted to monitoring flow of information with a few exceptions. Our approach shows its interest to accurately describe what the paths of executions that are not covered by the hooks LSM . In addition, the analysis is reproducible, allowing to follow the evolution of the kernel. We have ourselves analyzed the versions 4.3 and 4.7 of the Linux kernel, in addition to our modified release? Ed from the results of the first analysis.

Our third contribution gave us the opportunity to address an algorithmic Haggard pro- and implementation in the Linux kernel. Having found and demonstrated the problem the monitors? Ow of information implemented with
LSM We have shown that this problem is due to competitive conditions, and can bring the well-known causal problems. In e? And the problem of the conditions of competition arises when two greeting impacting the same information container are not observed in the causal order they are e? Ectués. This makes the greeting indirect composed of two vows is not considered because the point of

view of the greeting monitor, they could not logically be held. We have designed and demonstrated a new hues propagation algorithm to solve this problem by calculating a overapproximation shades if more of ux? Generation orders are possible and that the monitor can not know the correct order. We proved the correction of this algorithm using Coq. We then implemented in the kernel. There is a sub-contribution to this. To manage the conditions of competition is a necessary condition to treat greeting ongoing and we have again shown how information already maintained by the kernel can be exploited by monitors vows to treat the case of greeting caused by projections? les and shared memories. Our implementation,

When asked whether the *framework* LSM is well suited to the implementation of the follow-up? ow of information, so we made an ambiguous answer. In chapter 5 , Have shown that the hooks LSM are observation points appro- asked in general, with some modi? cations. However, when we proposed our own monitor in Chapter 6 We change? Ed the structure

framework to add hooks to? No system calls. The modi? Cations are not very significant in terms of number of lines of code modi? Ed but they highlight the limits of? Exibility and genericity LSM . Ambitious work extends this view deserve to be deepened, starting with a semantic system calls in terms of greeting information, as formal as possible. In e? And we spoke at the Chapter 2 that the nucleus is like a machine to make vows, the system calls are directed. However, unlike a processor or language, there is no clear semantics and disappointed nal system calls, especially those specific to Linux, but only a set of standards and rules expressed in natural language and pre- sented in the kernel documentation. This would increase the semantic con? Dence that we can have in the security mechanisms implemented with LSM and the scope of analysis as those we have developed.

? Finally, we have produced a Schedule contribution in this thesis has, in our view, very important: the use of the compiler to produce static analyzes and visualizations. Whether to understand the organization of a code base to explore paths of execution of a function, to implement a static analysis debugging a piece of code or to understand the operation a particular phase of the compilation process, the compilation of artifacts and compiler data structures can be made pro? t.

One problem we can not however be treated as part of a single thesis, regardless of the time we grant it, is that the adoption of the control flow of information as a security means in systems exploitation. In e? And, at present, control of greeting is far less prevalent than access control. In the Linux kernel *vanilla* , there are four security modules implementing a mandatory access control: AppArmor, SELinux, Smack and Tomoyo, but no ow control module?. In other areas such as dynamic analysis of malware, control of greeting however, already has demonstrated his interest. The problems we have dealt with are more relevant than ever.

Annex A

List of LSM hooks in version 4.7 of the kernel

A.1 hooks present in the nucleus where? Sky

The following table exhaustively list all hooks LSM available in version 4.7 of the Linux kernel *vanilla* , grouped by the data structures they protect. Underlined brackets are those used in the implementation of Rfblare.

A.1.1 Control of requests made to *binder* Android

binder_set_context_mgr Called to control the change of manager
of *binder*.

binder_transaction Called to control a call via the *binder* a process
to another.

binder_transfer_binder Called for controlling the transmission of a reference
on the *binder* from one process to another.

binder_transfer_file Called to control the transmission of a? Le via
***binder* from one process to another.**

A.1.2 Control calls ptrace

ptrace_access_check Called when a thread e? Ectue a system call ptrace
on a thread of another process for controlling the operation requested.

PTRACE_TRACEME Called when a thread requests a thread to another process
to trace it to control this operation.

A.1.3 Management and request thread permissions (capabilities)

capget Called when a thread calls capget so that the module returns the
capabilities of a thread.

capset Called when a thread calls capset for the change module? e the
capabilities of a thread.

able Called when veri? Cation that a thread has a certain capacity,
so that the module returns the response.

A.1.4 Resource Management quotas

quotactl Called to control the call for quota management functions
resources for users and processes.

quota_on Called to monitor compliance with the quotas on? Le.

A.1.5 Managing Access to the kernel log

syslog Called to control access by a thread to the kernel log (buffer
where the circular core print its newspapers).

A.1.6 Control handling the system clock

settime Called to control modi? Cation by the time thread and Date
current system.

A.1.7 allocation control of a new moire projection enmé-

vm_enough_memory Called in various locations when a new area of the core
Virtual memory must be allocated to a process,? n to control memory consumption.

A.1.8 Control and launch of managing a new executable

bprm_set_creds Called when loading a new program (with the call
system execve) to reset the permissions of the process according to the program.

bprm_check_security Called just before looking for an interpreter or
appropriate program for a certain type of binary loading of a new program to control the current operation or update the permissions process.

bprm_secureexec Called when loading a program to determine whether a
secure execution is required (which changes the behavior of the standard library, among other things).

bprm_committing_creds Called the? N loading a new program
to prepare the process just before its permissions are updated with those made earlier by previous hooks.

bprm_committed_creds Called the? N loading a new program,
Once the permissions process are ready to prepare the process for its implementation with its new permissions.

A.1.9 Managing security structures of the superblocks sys- tem? Virtual files

sb_alloc_security Called during the initialization of a superblock (e.g. during
mounting a system? files) to allocate and populate its security structure.

sb_free_security Called when the destruction of a superblock (eg when
the dismantling of a system? files) to deallocate its security structure.

sb_set_mnt_opts Called when updating security options of a super-
block.

sb_clone_mnt_opts Called when security options must be copied
a super block to another.

sb_parse_opts_str Called to initialize the security options of a superblock
from a string.

sb_copy_data Called to copy mounting options and a superblock
modi? er before they are passed to the system? file for editing.

sb_show_options Called to serialize mounting options of a superblock
in a string.

A.1.10 superblocks control operations of the virtual filesystem

sb_remount Called to control a winding operation of a system of
les.

sb_mount Called to control a mounting operation of a system? Les.
The function attached to a hook LSM can veri? er including dismantling spot and requested options.

sb_umount Called to control a disassembly operation of a system of
les.

sb_kern_mount Called just before the initialization of the superblock, during MON
floor, to initialize the security and check if the mounting structure can continue.

sb_statfs Called to control statistics on the recovery system
les superblock.

sb_pivotroot Called to control a general root-change operation
System? les, and possibly update some security structures in the process.

A.1.11 Managing security fields directory entries

dentry_init_security Called to populate the security of an entrance field
directory that has no associated inode. This hook is useful for system? NFS files.

d_instantiate Called to initialize the security field of an entry of reper-
tory being installed in the cache.

A.1.12 Operations Control on the paths of the virtual filesystem

Optional Compilation controlled by the option CONFIG_SECURITY_PATH. path_unlink Called to control an operation to delete a? Le (call system unlink) path_mkdir Called to control a directory creation operation (appeal system mkdir) path_rmdir Called to control a delete operation of a directory (System call rmdir) path_mknod Called to control the creation of a? Special file or pipe (System call mknod) path_truncate Called to check truncation or extension of a? Le (System call truncate) path_symlink Called to control the creation of a symbolic link (system call symlink) path_link Called to control the creation of a hard link (system call link) path_rename Called to control the renaming of a? Le (system call rename) path_chmod Called to control the mode change discretionary access a? le (system call chmod) path_chown Called to control the change of owner or group pro--owner of a? le (system call chown) path_chroot Called to control designed ment of a process in a nou-velle root system? les (system call chroot)

A.1.13 Management of i-nodes security system fields of? Virtual files

inode_alloc_security Invoked when i-node is created to allocate its structure of security.

inode_free_security Invoked when i-node is destroyed to deallocate its security structure.

inode_init_security Called to install the extended attributes of safety a brand new i-node.

A.1.14 Operations control the i-nodes of the virtual filesystem

inode_create Called to control the creation of a new? Le simple.

inode_link Called to control the creation of a hard link to a? Le.

inode_unlink Called to control the removal of a link to a? Le.

inode_symlink Called to control the creation of a symbolic link to a take a dump.

inode_mkdir Called to control the creation of a directory.

inode_rmdir Called to control the removal of a directory.

inode_mknod Called to control the creation of a? Special file or a pipe.

inode_rename Called to control the renaming of a? Le.

inode_readlink Called to control playback of a symlink to
know the destination.

inode_follow_link Called to control dereference symbolized a link
lic to keep the chippers? rage of a path.

inode_permission Called to control all operations on the i-nodes
other than those relating to the creation or destruction (eg, opening a? le).

inode_setattr Called to control modi? Cation of attributes of an inode
(Such as date of last access, for example);

inode_getattr Called to control playback of the attributes of an i-node.

inode_setxattr Called to control modi? Cation of extended attributes of a
inode, the security attributes. The function attached to this hook is not supposed to write its own
security attributes but only veri? Er the validity of the name and the value of the attribute.

inode_post_setxattr Called to update security structures of the i-
node and the directory entry after the modi? cation of extended attributes of an i-node.

inode_getxattr Called to control playback of an extended attribute of an inode
(Which can be a security attribute).

inode_listxattr Called to list the extended attributes of an i-node.

inode_removexattr Called to control the deletion of an extended attribute of a
i-node.

inode_need_killpriv Invoked when i-node is changed? Ed as to whether it is
necessary to remove the bit setuid and other special laws accordingly.

inode_killpriv Called when the bit setuid an i-node is removed has? n that
the security module adapted to turn the security context of the i-node accordingly.

inode_getsecurity Called to serialize a security attribute of the i-node in
a character string.

inode_setsecurity Called to control and e? Ectuer modi? Cation of an attribute
security of an i-node when the system? le does not support extended attributes. If the system? Les
manages, the pair of hooks inode_setxattr
and inode_post_setxattr is used instead.

inode_listsecurity Called to list the security attributes of an i-node in
a string when the system? le does not support extended attributes. If the system? Le handles, inode_listxattr
is used instead.

inode_getsecid Called to return the identi? Security ant an i-node.

inode_notifysecctx Called to indicate the security module what is the
security environment expected for an i-node. This hook is used by NFS to build di? ERA its
corresponding i-nodes to? Les stored on a remote server.

inode_setsecctx Called to position the security context of an i-node, and
affect this change in the extended attributes. This hook is used by NFS.

inode_getsecctx Called to retrieve the security context of an i-node.

A.1.15 Managing security fields descriptors - files

file_alloc_security Called to allocate and populate the security structure of a
descriptor? le.

file_free_security Called to deallocate the security structure of a descrip-
tor of? le.

file_set_fowner Called to position the value of the owner of a? Le,
so that the hook function associated file_send_sigiotask can the utili- ser.

file_open Called to store permissions to open the? Le, for power
cache security decisions.

A.1.16 Operations Control on descriptors? Le

file_permission Called to control an operation on a descriptor? Le,
essentially read and write or derivatives thereof.

file_ioctl Called to control an operation ioctl a? le.

file_mprotect Called to control the change of the protection of projections
tion in memory of a? le.

file_lock Called to control the use of locking a? Le.

file_fcntl Called to control an operation fcntl a? le.

file_send_sigiotask Named for veri? Er that the owner of a? Le is in
able to send the signal SIGIO or SIGURG. These signals are used on sockets to signal the process
owner that the socket has completed an input-output operation and is ready for a new operation.

file_receive Called to control the receipt of a descriptor? Le from
another process via a socket.

A.1.17 control the creation of a new projection moire enmé-

vm_enough_memory Called to control the allocation of a new zone of Me-
memory within the address space of a process, in case the security module wishes to enforce
quotas.

mmap_addr Called to control the creation of a new memory mapping
at a given address.

mmap_file Called to control memory in the screening of a? Le regular
existing.

A.1.18 Managing thread safety fields

task_free Called for the destruction of a thread to deallocate the associated memory.

cred_alloc_blank Called process for allocating a safety structure
Virgin.

cred_free Called to deallocate a process safety structure.

cred_prepare Allocate a process to initialize security structure from
another, by copying.

cred_transfer Called to initialize a process safety structure
from another, transferring the contents of the structure.

task_fix_setuid Called to update security structures associated with
a thread after a change UID, GID, etc.

task_to_inode Called to de? Ne the security structure of an i-node created by
a thread, especially for? special files describing the thread in
/ Proc / <pid>.

A.1.19 Operations Control involving threading or pro- cess

task_create Called to control the creation of a thread.

task_setpgid Called to control the change of a group number
thread.

task_getpgid Called to control the consultation of a group number
thread.

task_getsid Called to control consultation session number one
thread.

task_getsecid Called to retrieve the identi? Ant safety of a thread.

task_setnice Called to control change *niceness (* priority) of a
process.

task_setioprio Called to control the change in value ioprio a
thread.

task_getioprio Called to control the value of consultation ioprio a
thread.

task_setrlimit Called to control the boundary change of a resource
a process.

task_setscheduler Called to control the change of the political ordon-
funding applicable to a thread.

task_getscheduler Called to check scheduling policy applied
cable to a thread.

task_movememory Called for controlling the movement of pages of a NUMA node
to another, and where applicable, of a process to another (system calls
move_pages and migrate_pages). task_kill Called to control the transmission signal
by a process.

task_wait Called to control the expectation of an event of a process? Ls.

task_prctl Called to control an operation prctl.

A.1.20 Handling of security attributes of processes

getprocattr Called to serialize the form of a string of
security structure of a thread and monitor its recovery.

setprocattr Called to control modi? Cation of the security structure of a
thread and e? ectuer (by extracting the value of a string).

A.1.21 Control and management of tasks performed by a kernel thread

kernel_act_as Called to change the security context of a kernel thread.

kernel_create_files_as Called to allocate a kernel thread context
safety of a specific inode? that.

kernel_read_file Called for controlling the reading by a kernel thread of a
le speci? ed by the user space.

kernel_post_read_file Called to control the content of a? Le speci? Ed by
userspace read by a kernel thread.

kernel_module_request Called to control the loading of a module via a
call modprobe (in user space).

A.1.22 Managing security fields CPI System V

msg_msg_alloc_security Called to allocate the safety structure of a mes-
wise to be sent via a? the messages (POSIX and System V).

msg_msg_free_security Called to deallocate the security structure of a mes-
wise.

msg_queue_alloc_security Called to allocate the security structure of a? The
System messages V.

msg_queue_free_security Called to deallocate the security structure of a
the System V messages

shm_alloc_security Called to allocate the safety of a memory structure
Shared System V.

shm_free_security Called to deallocate the safety of a memory structure
Shared System V.

sem_alloc_security Called to allocate the safety structure of a group of
System V semaphores

sem_free_security Called to deallocate the security structure of a group
System V semaphores

A.1.23 Control CPI System V

ipc_permission Called to control the call to an operation on one of CPI
System V.

ipc_getsecid Called to retrieve the security context of an CPI System V.

msg_queue_associate Called to control the recovery identi? Ing a
the pre-existing System V messages.

msg_queue_msgctl Called to control an operation on the one of?
System messages V.

msg_queue_msgsnd Called to control sending a message via a? Of mes-
wise System V.

msg_queue_msgrcv Called for controlling the reception of a message from a
System message V.

shm_associate Called to control the recovery identi? Ant to unemémoire
Shared System V.

shm_shmctl Called to control an operation shmctl handling a
System V shared memory

shm_shmat Called for control of a shared memory System attachment V
in the address space of a process.

sem_associate Called to control the recovery of an identi? Group of ant
System V semaphore

sem_semctl Called to control an operation semctl for handling a
Signaling System Group V.

sem_semop Called to control an operation of the members of a group
System V semaphores

A.1.24 Managing Netlink message security fields and control of their issue

netlink_send Called upon issuance of a Netlink message control
transmission and for attaching a safety value to the message.

A.1.25 security contexts Management

ismaclabel Named for veri? Er if the value of a drunk? Er corresponds to a context
valid security.

secid_to_secctx Called to convert an identi? Ant (probably opaque) in
Complete security context.

secctx_to_secid Called for an identi? Ant safety from a context
of security.

release_secctx Called to deallocate a security context.

inode_invalidate_secctx Called to invalidate the security context of an
i-node and force his rehabilitation from the extended attributes or otherwise.

A.1.26 Operations Control on UNIX sockets

conditional compilation controlled by CONFIG_SECURITY_NETWORK. unix_stream_connect Called to

control the establishment of a connection
kind greeting between two UNIX domain sockets.

unix_may_send Called to control the connection and datagram transmission
on a UNIX socket type.

A.1.27 socket operations Control

conditional compilation controlled by CONFIG_SECURITY_NETWORK. socket_create Called to control the creation of a socket.

socket_post_create Named after the creation of a socket for adding at- Additional safety tributes to the socket. The sockets use as dedicated security structure that their i-node and a security structure.

socket_bind Called to control the attachment of a socket to an address.

socket_connect Called to check the connection to a socket.

socket_listen Called to control the attachment of a socket to a port and start listening.

socket_accept Called to control the acceptance of a connection to a socket.

socket_sendmsg Called to control the transmission of a message on a socket.

socket_recvmsg Called to control the receipt of a message from a socket.

socket_getsockname Called to control the recovery of the local address the socket.

socket_getpeername Called to control the recovery of the address of the peer remote from the socket.

socket_getsockopt Called to control the recovery options positioned on the socket.

socket_setsockopt Called to control the handling of the socket options.

socket_shutdown Called to control the closing of a socket.

socket_sock_rcv_skb Called for controlling the reception of a packet on a INET domain socket before passing it in? Iter network.

A.1.28 Managing sockets security fields

conditional compilation controlled by CONFIG_SECURITY_NETWORK. socket_getpeersec_stream Called to recover (and control for the recovery tion) security structure associated with the remote peer of a mode socket greeting (INET, using TCP or UNIX).

socket_getpeersec_dgram Called to recover (and monitor recovery) safety structure associated with a message from a datagram socket (INET, so using UDP).

sk_alloc_security Called to allocate a specific security structure? That the sockets.

sk_free_security Called to deallocate the specific security structure? That the sockets.

sk_clone_security Called to copy a specific security structure? That the sockets.

sk_getsecid Called to retrieve the identi? Ant security associated with a socket.

sock_graft Called to position identi? Ant safety of the i-node of the socket to identi? ant safety of the socket itself.

inet_conn_request Called upon receiving a connection request for
update identi? connection request the security ant from that of the server socket and identi? ing security of the remote peer.

inet_csk_clone Called for the establishment of a session to update the identical
ant safety socket? lle from the identi? ant safety of the connection request.

inet_conn_established Named after the establishment of a session to make
day identi? ant security associated with the remote peer in the socket (server side), according to the identi? ant security associated with the received packet.

secmark_relabel_packet Call for controlling a relabeling operation
a package by a user process.

secmark_refcount_inc Called to report to the security module a rule
additional re-labeling was loaded.

secmark_refcount_dec Called to report to the security module a rule
re-labeling was discharged.

req_classify_flow Called to update the identi? Security ant a greeting
Internet after identi? ant safety of the connection request.

A.1.29 Managing security fields virtual network interfaces

conditional compilation controlled by CONFIG_SECURITY_NETWORK. tun_dev_alloc_security Called
for allocating a safety structure for a
TUN interface.

tun_dev_free_security Called to deallocate the security structure of a
TUN interface.

A.1.30 Control of operations on the network interfaces vir- tual

conditional compilation controlled by CONFIG_SECURITY_NETWORK. tun_dev_create Called to
control the creation of a new TUN interface.

tun_dev_attach_queue Called to control an attachment operation de-
mended by the current process to a TUN interface.

tun_dev_attach Called to update the security context associated with a
socket when attached to a TUN interface.

tun_dev_open Called for controlling an opening of an interface operation
TUN.

A.1.31 Managing security fields XFRM

conditional compilation controlled by CONFIG_SECURITY_NETWORK_XFRM.
XFRM is a network packet processing framework, it is used for IPSec implemented.

xfrm_policy_alloc_security Called to control the creation of a new
policy in the basic policies and allocate a security structure.

xfrm_policy_clone_security Called to allocate a new structure se-
curity for political and boot from another.

xfrm_policy_free_security Called to deallocate a security structure
policy.

xfrm_policy_delete_security Called to control the destruction of a poli-
tick the base.

xfrm_state_alloc Called to control the creation of a new association
in the database used by XFRM and allocate the corresponding security structure. The structure is initialized from a context manufactured by a suitable user-space program.

xfrm_state_alloc_acquire Similar to xfrm_state_alloc but the context of
security with which the new structure must be initialized is speci? ed by an identi? Security ant.

xfrm_state_free_security Called to deallocate the security structure of a
association.

xfrm_state_delete_security Called to control the destruction of asso-
ciation.

A.1.32 Operations Control on XFRM policies

conditional compilation controlled by CONFIG_SECURITY_NETWORK_XFRM. xfrm_policy_lookup Called

to monitor the implementation of policies XFRM
a greeting network.

xfrm_state_pol_flow_match Called to see if a given policy ap-
plique a greeting network.

xfrm_decode_session Called to manage all packages of the same
session well use the same identi? ant safety.

A.1.33 Managing security fields kernel Key Directory

conditional compilation controlled by CONFIG_KEYS. key_alloc Called to control the creation of a new

key in the
keys and allocate a security structure.

key_free Called to deallocate the security structure associated with a key.

A.1.34 control operations e? Ectuées the core Key Directory

conditional compilation controlled by CONFIG_KEYS. key_permission Called to control an operation on

a key register
key.

key_getsecurity Called to serialize into a string context
security associated with a key.

A.1.35 Managing security fields of the kernel auditing system

conditional compilation controlled by CONFIG_AUDIT. audit_rule_init Called to initialize the safety structure associated with a
new audit rule.
audit_rule_free Called when the audit rule is discharged, a? N deallocate
the corresponding security structure.

A.1.36 control of the core to said system operations

conditional compilation controlled by CONFIG_AUDIT. audit_rule_known Called to see if a given audit rule contains
fields related to the security module.
audit_rule_match Called to see if a given security context matches
a satisfactory rule veri? cation made by audit_rule_known.

A.2 hooks added to Rfblare

Rfblare requires some hooks LSM Further as detailed in Chapters 5 and 6 . They are listed in the table below.

A.2.1 Control? N of a system call causing a greeting information

syscall_before_return This hook is called when returning from a system call
causing a greeting as discreet read, write, and when a sys- tem call is interrupted by a signal.

A.2.2 Managing security fields address spaces

mm_dup_security This hook is called when a call to clone or fork for
duplicate the security structure attached to the address space of the calling process.

mm_sec_free This hook decrements the reference count of the se- structure
rity of the address space of the calling process and deallocates if necessary. It is called when a thread disappears.

A.2.3 Control of operations on? The POSIX Message

mq_store_msg This hook is called to control receiving from aMessage
a? the POSIX message.

No additional hook is required to issue because the kernel uses the same interface, so the same brackets for the allocation of posts? System V and POSIX.

A.2.4 Additional control ptrace

ptrace_unlink This hook is called when a thread stops ptracer another.

Appendix B

De? Nition of abstract semantics for our static analysis and proof correction

In this printable version of this script, the script complete Rooster the correctness proof of the abstract semantics necessary for static analysis described in Chapter 5 is not included. It is available at the following address: https: //kayrebt.gforge.inria.fr/proofs.html .

Appendix C

Formal description of the spread of hues Rfblare and proof correction

In this printable version of this script, the script complete Rooster giving the formal description of the colors propagation algorithm Rfblare and its correctness proof is not included. It is available at the following address:
https://blare-ids.org/rfblare/ .

Bibliography

[1] A. Abraham, R. Andriatsimandefitra, A. Brunelat, JF Lalande and Viet V. Triem Tong. "GroddDroid a gorilla for triggering malicious Behaviors". In: International Conference on Malicious and Unwanted Software (MALWARE 2015). Fajardo, Puerto Rico: IEEE, Oct. 2015, p.. 119-127. isbn: 978-1-5090-0317-4. doi: 10.1109 / MALWARE.2015.7413692 (cf. p. 22).

[2] Radoniaina Andriatsimandefitra Valerie Tong Viet Triem and Thomas Saliou. *Information Flow Policies vs. Malware*. postponement. September 16, 2013 (see p. 22 , 77).

[3] Arnd Bergmann. [*PATCH 20/20] BKL: That's all, folks [LWN.net]*. E-mail published on the Linux kernel development list. January 25. 2011. url: https: //lwn.net/Articles/424677/ (visited 05.07.2017).

[4] Dirk Beyer, Adam J. Chlipala, Thomas A. Henzinger, Ranjit Jhala and Rupak Majumdar. "The Blast Query Language for Software Veri? Cation." In: International Static Analysis Symposium (SAS 2004). Ed. Roberto Giacobazzi. T. 3148. Lecture Notes in Computer Science. Verona, Italy: Sprin- ger, August 2004, p. 2-18. isbn: 978-3-540-27864-1. doi: 10.1007 / 978-3-54027864-1_2 (cf. p. 30 , 126).

[5] Dirk Beyer and Alexander K. Petrenko. "Linux Driver Veri? Cation." in: International Symposium On Leveraging Applications of Formal Methods, Veri? Cation and Validation (Isola 2012). Ed. Tiziana Margaria and Bernhard Steffen. T. 7610. Lecture Notes in Computer Science. Heraklion, Greece: Springer, Oct. 2012, p.. 1-6. isbn: 978-3-642-34031-4. doi: 10.1007 / 978-3-642-34032-1 (cf. p. 32).

[6] Daniel P. Bovet and Marco Cesati. *Understanding the Linux Kernel*. 3 e ed. O'Reilly Media, in November 2005. 944 p. ISBN: 978-0-596-00565-8 (cf. p. 35 , 55).

[7] Neil Brown. *Sparse: a look under the hood*. Linux Weekly News. June 8, 2016. url: https://lwn.net/Articles/689907/ (visited 02/01/2017) (see p. 29).

[8] TY Chen, H. Leung and IK Mak. "Adaptive Random Testing". in: *Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making*. Asian Com- puting Science Conference (ASIAN 2004). Ed. Michael J. Maher. T. 3321. Lecture Notes in Computer Science. DOI: 10.1007 / 978-3-540-30502- 6_23. Chiang Mai, Thailand: Springer, 2004, p. 320-329. isbn: 978-3-540-24087-7. doi: 10.1007 / 978-3-540-30502-6_23 (cf. p. 31).

[9] Yang Chen, Alex Groce, Chaoqiang Zhang, Weng-Keen Wong Xiaoli Fern, Eric Eide and John Regehr. "Taming Compile fuzzers." In ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2013). Seattle, WA, USA: ACM, 2013, p. 197-208. isbn: 978-1-4503-2014-6. doi: 10.1145 / 2491956.2462173 (cf. p. 31).

[10] Winnie Cheng, RK Dan Ports, David Schultz, Victoria Popic Aaron Blankstein, James Cowling, Dorothy Curtis, Liuba Shrira and Barbara Liskov. "Abstractions for Usable Information Flow Control in Aeolus'. In: USENIX Annual Technical Conference (USENIX ATC 2012). Boston, MA, USA: USENIX Association, 2012, p. 139-151 (p. 11 - 13 , 17).

[11] Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem and Dawson Engler. "An Empirical Study of Operating Systems Errors". In ACM Symposium on Operating Systems Principles (SOSP 2001). Ban ?, Alberta, Canada: ACM, 2001 p. 73-88. isbn: 978-1-58113-389-9. doi: 10.1145 / 502034.502042 (cf. p. 35).

[12] Kees Cook. *gcc-plugins: Initial Add x86_64 KERNEXEC plugin.* E-mail published on the Linux kernel development list. January 13 2017. url: https: //lwn.net/Articles/711655/ (visited 02.06.2017). [13] Kees Cook. *gcc-plugins: Add structleak for more stack initialization.* E-mail published on the Linux kernel development list. January 13 2017. url: https: //lwn.net/Articles/711692/ (visited 02.06.2017). [14] Jonathan Corbet. *Kernel building with GCC plugins.* Linux Weekly News. 14 June 2014. url: https: / / LWN. net / articles / 691102 / (visited 06/02/2017) (see p. 30).

[15] Jonathan Corbet, Jake Edge and Rebecca Sobol. *LWN.net News from the source.* Linux Weekly News. 1998. url: https://lwn.net (cf. p. 35).

[16] Maximiliano Cristiá and Pablo Enrique Mata. "Runtime enforcement of noninterference by duplicating Processes and their memories. " In: Workshop Seguridad Informática (WSegI 2009). T. 00008. 2009 Mar del Plata, Argentina, in August 2009 (see p. 11 , 77).

[17] Maximiliano CRISTIA and Pablo Enrique Mata. *Flowx: No. of Implementacion interferencia Linux.* Rosario, Argentina: Facultad de Ciencias Exactas, engi- niería Agrimensura there, Universidad Nacional de Rosario, 27 November 2009, p.. 186 (cf. p. 13 , 77).

[18] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman and F. Kenneth Zadeck. "E? Ciently Computing Static Single Assignment Form and the Control Dependence Graph". in: *ACM Transactions on Programming Languages and Systems* 13.4 (October 1991), p. 451-490. issn: 0164-0925. doi: 10.1145 / 115372.115320 (cf. p. 62 , 65).

[19] Al Danial. *CLOC - Count Lines of Code.* 00021. July 30, 2014 (p. 35). [20] Dorothy E. Denning. "A Lattice Model of Secure Information Flow." in: *Communications of the ACM* 19.5 (May 1976), p. 236-243. issn: 0001-0782. doi: 10.1145 / 360051.360056 (cf. p. 7 , 13).

[21] Elizabeth Dorothy Denning Robling and Peter J. Denning. "Certi? Cation of programs for secure information? ow ". in: *Communications of the ACM* 20.7 (July 1977). Ed. Robert L. Ashenhurst, p. 504-513. issn: 0001-0782 (p. 13).

[22] David Drysdale. *Coverage-guided kernel fuzzing with syzkaller.* Linux Weekly News. 2 March 2016. url.: https://lwn.net/Articles/677764/ (visited 06/02/2017) (see p. 31).

[23] Bruno Dutertre and Leonardo Moura. *The Yices SMT solver.* SRI Internal, 2006 (see p. 123).

[24] Jake Edge. *A pair of GCC plugins.* Linux Weekly News. January 27 2017. url: https://lwn.net/Articles/712161/ (visited 06/02/2017) (see p. 30).

[25] Jake Edge. *The kernel address sanitizer.* Linux Weekly News. 14 September 2014. url.: https://lwn.net/Articles/612153/ (visited 02/02/2017) (see p. 32).

[26] Antony Edwards and Trent Jaeger. "Maintaining the correctness of the Linux security framework modules. " in: *Ottawa Linux Symposium.* 2002, p. 223 (cf. p. 33).

[27] J. Ellson, ER Gansner, E. Koutsofios, SC North and G. Woodhull. "Graphviz and Dynagraph - Static and Dynamic Graph Drawing Tools. " in: *Graph Drawing Software.* Ed. Mr. Junger and P. Mutzel. Berlin / Heidelberg, Germany: Springer, 2004, p. 127-148. ISBN: 3-540-00881-0 (see p. 66).

[28] William Enck, Peter Gilbert, Byung-Gon Chun, P. Landon Cox, Jaeyeon Jung and Patrick McDaniel Anmol N. Sheth. "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring is smartphones." In: USENIX Conference on Operating Systems Design and Implementation (OSDI 2010). Vancouver, BC, Canada: USENIX Association, 2010, p. 1-6 (p. 12 , 13 , 77).

[29] JS Fenton. "Memoryless Subsystems". in: *The Computer Journal* 17.2 (1st Jan. 1974), p. 143-147. issn: 0010-4620. doi: 10.1093 / comjnl / 17.2.143 (See p. 10).

[30] I? Rey Scott Foster. "Type quali ers: lightweight speci cations to Improve? software quality. " Thesis doct. Berkeley, CA: University of California at Berkeley, 2002 (see p. 29).

[31] Tal Garfinkel. "Traps and Pitfalls: Practical Problems in System Call Inter Position Based Security Tools. "In: Network and Distributed Systems Security Symposium (NDSS 2003). San Diego, CA, USA: The Internet Society, February 2003 p. 163-176. ISBN: 1-891562-16-9 (cf. p. 24 , 26 , 27).

[32] Samir Genaim and Fausto Spoto. "Information Flow Analysis for Java Bytecode. " In: International Conference on Veri cation, Model Checking, and Abstract Interpretation (VMCAI 2005)?. Ed. Radhia Cousot. T. 3385. Lecture Notes in Computer Science. Paris, France: Springer, January 2005, p. 346-362. isbn: 978-3-540-24297-0. doi: 10.1007 / 978-3-540-30579-8_23 (See p. 11).

[33] George Lawrence, Valerie Tong Viet Triem and Ludovic Mé. "Blare Tools: A Policy-Based Intrusion Detection System Automatically Set by the Security Policy. " In: International Workshop on Recent Advances in Intrusion Detection (RAID 2009). Ed. Engin Kirda, Somesh Jha and Davide Balzarotti. T. 5758. Lecture Notes in Computer Science. Saint-Malo, France: Springer, September 2009, p.. 355-356. isbn: 978-3-642-04341-3. doi: 10.1007 / 978-3-64204342-0_22 (cf. p. 11 , 13 , 21 , 73 , 77 , 129).

[34] Lawrence Georget, Mathieu Jaume, Piolle William Frederick Tronel and Valérie Viet Triem Tong. "Information Flow Tracking System for Linux Handling Concurrent Calls and Shared Memory." in: *Software Engineering and Formal Methods.* Software Engineering and Formal Methods (MPAC 2017). Ed. Alessandro Cimatti and Marjan Sirjani. T. 10469. Lecture Notes in Computer Science. Trento, Italy: Springer, Cham, Switzerland, September 4, 2017,. p. 1-16. isbn: 978-3-319-66196-4. doi: 10.1007 / 978-3-319-66197-1_1 (See p. 148).

[35] Laurent Georget, Mathieu Jaume Piolle Guillaume, Frédéric and Tronel Valérie Viet Triem Tong. "Monitoring? Ow proper information on Linux". In: Formal Approaches in Software Development Assistance (AFADL 2017). Ed. Akram Idani and Nikolai Kosmatov. Montpellier, France, in June 2017.

[36] Laurent Georget, Mathieu Jaume Piolle Guillaume, Frédéric and Tronel Valérie Viet Triem Tong. "Verifying the Reliability of Operating System- Level Information Flow Control Systems in Linux". In: FME Workshop on Formal Methods in Software Engineering (formalized in 2017). Buenos Aires, Argentina: IEEE Press, May 2017, p. 10-16. isbn: 978-1-5386-0422-9. doi: 10.1109 / FormaliSE.2017..1 (cf. p. 126).

[37] Laurent Georget, Frederic Tronel and Valérie Viet Triem Tong. "Kayrebt: An Activity Diagram Extraction and Visualization Toolset Designed for the Linux codebase. " In: IEEE Working Conference on Software Visualization (VISSOFT 2015). Bremen, Germany: IEEE, September 2015, p.. 170-174. doi: 10. 1109 / VISSOFT.2015.7332431 (cf. p. 71).

[38] Christopher Hauser. "Intrusion detection systems distributed by hue spread kernel level ". Doctoral thesis. Rennes, France: University of Rennes 1, June 2013 (see p. 11 , 20 , 21 , 75 , 82 , 132 , 133).

[39] Christopher Hauser and William Brogi. *KBlare*. 2014. Software. url: https: //git.blare-ids.org (visited 04.10.2017).

[40] William Hiet. "Intrusion detection set by secu- policy rity through collaborative control flow of information within the operating system and applications: Linux implementation for Java pro- grams. " Thesis doct. Rennes, France: Supélec, 2008 (see p. 11 , 20 , 21 , 77).

[41] D. Richard Hipp, Dan Kennedy and Joe Mistachkin. *SQLite.* Version 3.18.0, published March 30, 2017. Software. url: https://www.sqlite.org (visited 05.07.2017).

[42] Trent Jaeger, Anthony Edwards and Xiaolan Zhang. "Consistency analysis of investment authorization hook in the Linux security module framework ". in: *ACM Trans. Inf. Syst. Secur.* 7.2 (2004), p. 175-205 (p. 32 , 150).

[43] Sushil Jajodia, Ravi S. Sandhu and Barbara T. Blaustein. "Solutions to the Polyinstanciation Problem ". in: *Information Security: An integrated Collection of Essays.* Ed. Marshall D. Abrams and Harold J. Sushil Jajodia Podell. Los Alamitos, CA: IEEE Computer Society Press, 1995, p. 493-530. ISBN: 0-8186-3662-9 (p. 15).

[44] Mathieu Jaume. "Semantic Comparison of Security Policies: From Access Policies to Flow Control Properties ". In: IEEE Symposium on Security and Privacy Workshops (SPW 2012). San Francisco, CA, USA: IEEE, May 2012, p. 60-67. isbn: 978-1-4673-2157-0. doi: 10.1109 / SPW.2012.33 (cf. p. 7).

[45] Mathieu Jaume, Radoniaina Andriatsimandefitra, Valérie Viet Triem Tong and Ludovic Mé. "Secure states versus Secure executions: From access control to control ow." In: International Conference on Information Systems Security (ICISS 2013). T. 8303. Lecture Notes in Computer Science. Calcutta, India: Springer, December 2013, p. 148-162. isbn: 978-3-642-45203-1. doi: 10.1007 / 9783-642-45204-8_11 (cf. p. 7).

[46] Mathieu Jaume, Valerie Tong Viet Triem and Ludovic Mé. "Flow based interpretation of access control: Detection of illegal information ows? ". In: International Conference on Information Systems Security (ICISS 2011). T. 7093. Lecture Notes in Computer Science. Kolkata, India: Springer, December 2011, p. 72-86. isbn: 978-3-642-25559-5. doi: 10.1007 / 978-3-642-25560-1_5 (cf. p. 7 , 8).

[47] Rob Johnson and David Wagner. "Finding User / Kernel Bugs with Pointer Type Inference ". In: USENIX Security Symposium (USENIX 2004). T. 13. San Diego, CA, USA: USENIX Association, 2004 (see p. 29).

[48] Dave Jones. *Trinity.* Version v1.7, published October 28, 2016. Software. url: https://github.com/kernelslacker/trinity (visited 02.06.2017). [49] Michael Kerrisk. *LCA: The Trinity fuzz testing.* Linux Weekly News. February 6 2013. url: https://lwn.net/Articles/536173/ (visited 06/02/2017) (see p. 31).

[50] Alexey Khoroshilov Vadim Mutilin Alexander Petrenko and Vladimir Zakharov. "Establishing Linux Driver Veri? Cation process." in: *Prospects of Informatics Systems.* Andrey Ershov Memorial International Conference on Perspectives of System Informatics. T. 5947. Lecture Notes in Computer Science. DOI: 10.1007 / 978-3-642-11486-1_14. Novosibirsk, Russia: Springer, June 2009 p. 165-176. isbn: 978-3-642-11485-4. doi: 10.1007 / 978-3-642-11486-1_14 (See p. 32).

[51] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer Marinus Frans Kaashoek Eddie Kohler and Robert Morris. "Information? Ow control for OS abstractions standard". In ACM Symposium on Operating Systems SIGOPS Principles (SOSP 2007). Stevenson, WA, USA: ACM, Oct. 2007, p.. 321-334. isbn: 978-1-59593-591-5. doi: 10.1145 / 1294261.1294293 (cf. p. 11 , 13 , 77).

[52] J. Leonard LaPadula and D. Elliott Bell. *Secure Computer Systems: A mathematical tical Model.* MTR-2547 (ESD-TR-73-278-II) Vol. 2. Bedford MITER Corp., May 1973 (see p. 6).

[53] Donald C. Latham. *Trusted Computer System Evaluation Criteria (Orange Book).* 5200.28-STD. Arlington County, VA, USA: Department of Defense, December 1985 p. 116 (cf. p. 14).

[54] Robert Love. *Linux Kernel Development.* 3 e ed. Upper Saddle River, NJ: Addison Wesley, July 2, 2010. 440 p. ISBN: 978-0-672-32946-3 (cf. p. 35 , 55).

[55] TF Lunt, DE Denning, RR Schell, Mr. Heckman and WR Shockley. "The SeaView security model. " in: *IEEE Transactions on Software Engineering* 16.6 (June 1990), p. 593-607. issn: 0098-5589. doi: 10.1109 / 32.55088 (cf. p. 15).

[56] M. Douglas McIlroy and James A. Reeds. "Multilevel Security in the UNIX Tradition. " in: *Software: Practice and Experience* 22.8 (1992), p. 673-694. issn: 0038-0644. doi: 10.1002/spe.4380220805 (cf. p. 10 , 13 , 14 , 140).

[57] Mr. Mendonça and N. Neves. "Fuzzing WiFi Drivers to Locate Security vulrabilities ". In: European Dependable Computing Conference (EDCC 2008). Khaunas, Lithuania: IEEE, May 2008, p. 110-119. isbn: 978-0-7695-3138-0. doi: 10.1109/EDCC-7.2008.22 (cf. p. 31).

[58] Jason Merrill. "GENERIC and Gimple: A new tree representation for Entire functions ". in: *GCC Developers Summit*. 2003, p. 171-180 (p. 62).

[59] Barton P. Miller, Louis Fredriksen and Bryan So. "An Empirical Study of the Reliability of Unix Utilities ". in: *Communications of the ACM* 33.12 (Dec. 1990), p. 32-44. issn: 0001-0782. doi: 10.1145/96267.96279 (cf. p. 31).

[60] Justin J. Miller. "Graph database applications and concepts with Neo4j." in: *Southern Association for Information Systems Conference Proceedings*. Southern Association for Information Systems Conference. T. 24. Atlanta, GA, USA: Association for Information Systems, March 2013 (see p.. 71).

[61] January Tobias Mühlberg and Gerald Lüttgen. "Blasting linux code." In: Inter National Workshop on Formal Methods for Industrial Critical Systems (FMICS 2006). Ed. Luboš Brim, Boudewijn Haverkort Martin Leucker and Jaco van de Pol. T. 4346. Lecture Notes in Computer Science. Bonn, Germany: Springer, 2006, p. 211-226. isbn: 978-3-540-70951-0. doi: 10.1007/978-3540-70952-7_14 (cf. p. 30).

[62] Toby Murray, Daniel Matichuk Matthew Brassil Peter Gammie, Timothy Bourke, Sean Seefried, Corey Lewis, Xin Gao and Gerwin Klein. "SEL4: From General Purpose to a Proof of Information Flow Enforcement". In: IEEE Symposium on Security and Privacy (S & P 2013). Berkeley, CA, USA: IEEE, May 2013, p. 415-429. isbn: 978-0-7695-4977-4. doi: 10.1109/SP.2013.35 (See p. 11).

[63] Divya Muthukumaran, Trent Jaeger and Vinod Ganapathy. "Leveraging "Choice" to automatic investment authorization hook ". In: 00006. ACM Press, 2012, p. 145. isbn: 978-1-4503-1651-4. doi: 10.1145/2382196.2382215 (See p. 33).

[64] Andrew C. Myers. "JFlow: Mostly Practical-static Information Flow Control." in: *ACM-SIGPLAN SIGACT Symposium on Principles of Programming Languages*. San Antonio, Texas, USA: ACM, 1999, p. 228-241. isbn: 1-58113-095-3. doi: 10.1145/292540.292561 (cf. p. 11 , 13 , 17).

[65] Andrew C. Myers and Barbara Liskov. "A Decentralized Model for Information Flow Control ". In ACM Symposium on Operating Systems Principles (SOSP 1997). Saint-Malo, France: ACM, 1997. 129-142. isbn: 978-0-89791-916-6. doi: 10.1145/268998.266669 (cf. p. 12 , 15 , 17).

[66] Andrew C. Myers and Barbara Liskov. "Protecting Privacy Using the decentralized lized Label Model ". in: *ACMTransactions on Software Engineering Methodology* 9.4 (October 2000), p. 410-442. issn: 1049-331X. doi: 10.1145/363516.363526 (See p. 11 - 13 , 17).

[67] Adwait Nadkarni, Benjamin Andow WilliamEnck and Somesh Jha. "Practical DIFC Enforcement on Android. " In: USENIX Security Symposium (USENIX 2016). Austin, TX, USA: USENIX Association, August 2016, p. 1119-1136. ISBN: 978-1-931971-32-4 (cf. p. 11 , 13 , 21 , 73 , 77 , 129).

[68] Amon Ott. *RSBAC and LSM.* RSBAC: Extending Linux Security Beyond the Limits. 2006. url: https://www.rsbac.org/documentation/why_rsbac_ does_not_use_lsm (visited 02/10/2017) (see p. 73).

[69] Yoann Padioleau Julia L. Lawall René Rydhof Hansen and Gilles Muller. "Documenting and Automating Collateral developments in Linux Device Dri to." In: European Conference on Computer Systems (EuroSys 2008). Glas- gow, Scotland ACM, April 2008, p.. 247-260 (p. 29).

[70] Perl 5 Porters, ed. *perlsec - Perl Manual.* Perl version 5 Version 24 May 2016 (See p. 10).

[71] Hendrik Post, Carsten Sinz and Wolfgang Küchlin. "Towards automatic software model checking of Thousands of Linux modules-a case study with Avi-nux ". in: *Software Testing, Veri? Cation and Reliability* 19.2 (2009), p. 155-172 (p. 32).

[72] François Pottier and Vincent Simonet. "Information Flow Inference for ML." in: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 25.1 (2003), p. 117-158. issn: 0164-0925. doi: 10.1145 / 596980.596983 (cf. p. 11).

[73] Q-Success. *Usage Statistics and Market Share of Linux for Websites, March 2017.* W3Techs Web Technology Surveys. 2016 URL: https: // w3techs. com / technology / details / os-linux / all / all (visited 03/14/2017) (see p. 36).

[74] Gordon Henry Rice. "Class of Recursively Enumerable Sets and Their decision Problems. " in: *Transactions of the American Mathematical Society* 74.2 (1953), p. 358-366. issn: 00029947. doi: 10.2307 / 1990888 (cf. p. 28 , 94).

[75] Indrajit Roy and Don Porter. *Laminar.* Version v2, published on 20 August 2014. Software. url: https: / / sourceforge. net / p / jikesrvm / research archive / 26 (visited 02.22.2017). [76]

Indrajit Roy, Donald E. Porter, Michael D. Bond, Kathryn S. McKinley and EmmettWitchel. "Laminar: Practical Fine-grainedDecentralized Information Flow Control." In ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2009). Dublin, Ireland: ACM, June 2009 p. 63-74. isbn: 978-1-60558-392-1. doi: 10.1145 / 1543135.1542484 (cf. p. 11 , 13 , 19 , 20 , 73 , 77 , 129).

[77] Paul Rusty Russell. "Unreliable guide to hacking the Linux kernel." 2000 (See p. 30 , 57).

[78] Andrey Ryabinin. *UBSan: run-time behavior unde ned sanity checker.* E-mail published on the Linux kernel development list. 20 October 2014. URL.: https: //lwn.net/Articles/617364/ (visited 02.02.2017).

[79] Jerome. H. Saltzer and Michael. D. Schroeder. "The Protection of Information in Computer Systems ". in: *Proceedings of the IEEE* 63.9 (September 1975), p. 1278-1308 issn: 0018-9219. doi: 10.1109 / PROC.1975.9939 (cf. p. 10).

[80] Randal L. Schwartz, Tom and Brian D. Foy Phoenix. *Learning Perl.* 6 e ed. O'Reilly Media, June 2011. 388 p. ISBN: 978-1-4493-0358-7 (cf. p. 10).

[81] PE Shved VS Mutilin and MU Mandrykin. "Experience of Improving the blast static veri? cation tool. " in: *Programming and Computer Software* 38.3 (1st June 2012), p. 134–142. issn: 0361-7688, 1608-3261. doi: 10.1134 / S0361768812030061 (cf. p. 30).

[82] KY Sim F.-C. Kuo and R. Merkel. "Fuzzing the Out-of-Memory Killer on Embedded Linux: An Adaptive Random Approach ". In ACM Symposium on Applied Computing (SAC 2011). TaiChung Taiwan: ACM, 2011, p. 387–392. isbn: 978-1-4503-0113-8. doi: 10.1145 / 1982185.1982268 (cf. p. 31).

[83] Jiří Slabý. "Automatic Bug-? Nding Techniques for Large Software Projects". Doctoral thesis. Brno, Czech Republic: Masaryk University, Faculty of Infor- matics, March 4, 2014 (see p.. 29).

[84] Stephen Smalley. [*PATCH 0/2] Addmissing LSMhooks inmq_timed {send, receive} and splice, Email on the Linux Security Modules development mailing list.* E-mail published on the Linux kernel development list. July 2016. url.: http://thread.gmane.org/gmane.linux.kernel.lsm/28737 .

[85] Stephen Smalley, Chris Vance and Wayne Salamon. *Implementing SELinux as a Linux security module.* 01-043. Santa Clara, CA, USA: Network Associates Inc., 2001, p. 139 (cf. p. 73).

[86] Brad Spender. "SSTIC 2016 Keynote - grsecurity". Safety Symposium of information technologies and communications. Rennes, France, 1st June 2006 (see p. 30).

[87] Brad Spender. *Why does not use LSM grsecurity?* grsecurity. 2008. url: https: //grsecurity.net/lsm.php (visited 02/10/2017) (see p. 73).

[88] Richard Matthew Stallman and the GCC developer community. *plugins - GNU Compiler Collection (GCC) Internals.* 2015 URL: https: //gcc.gnu. org / onlinedocs / gccint / Plugins.html # Plugins (visited 05/18/2015) (see p. 61).

[89] Richard Matthew Stallman and the GCC developer community. *using the GNU Compiler Collection (GCC).* 2013 (see p. 30 , 61 , 190).

[90] StatCounter. *StatCounter Global Stats - Browser, OS, Search Engine Including Use Mobile Share.* StatCounter Global Stats. Nov. 2016. url: http: // gs. statcounter.com/ (visited 03/14/2017) (see p. 36).

[91] Erich Strohmaier, Jack Dongarra Horst Simon and Martin Meuer. *Operating Family system / Linux.* TOP500 Supercomputer Sites. Nov. 2016. url: https: //www.top500.org/statistics/details/osfam/1 (visited 03/14/2017) (see p. 36).

[92] Technical Committee X3J11. *American National Standard for Information Systems - Programming Language C.* X3.159-1989. American National Standards Institute, 1989 (see p. 57).

[93] The CoqDevelopment Team. *The Coq Proof Assistant Reference Manual.* December 14 2016 (see p. 88 , 129).

[94] Craig Timberg. "The kernel of the argument." in: *Washington Post (* November 5 2015) (p. 36).

[95] Linus Torvalds. *Re: [Regression w / patch] Digital commit causes to user space misbahave (was: Re: Linux 3.8-rc1).* E-mail published on the Linux kernel development list. December 23 2012. url: https://lkml.org/lkml/2012/12/23/ 75 (visited 02.28.2017). [96] Linus Torvalds. *Sparse "context" checking* .. E-mail published on the devel- list

Linux kernel opment. 30 October 2004. URL.: https://lwn.net/Articles/ 109066 / (visited 02.03.2017).

[97] Linus Torvalds. *The Linux Kernel.* Version 4.7, published July 24, 2016. Soft-

sky. Linux Kernel Organization, Inc. url: https://kernel.org (visited 02.02.2017).

[98] Linus Torvalds, Josh Triplett and Christopher Li. *Sparse - a semantic parser for C.* 2003. Software. url: https://sparse.wiki.kernel.org .

[99] Neil Vachharajani, Matthew J. Bridges, Jonathan Chang, Ram Rangan, Guilherme Ottoni, Jason A. Blome, George A. Reis, Manish Vachharajani and David I. August. "RIFLE: An Architectural Framework for User-Centric Information Flow Security". In: International Symposium on Microarchi- tecture (MIC 2004). Portland, OR, USA: IEEE, 2004, p. 243-254. isbn: 0-7695-2126-6. doi: 10.1109 / MICRO.2004.31 (cf. p. 11).

[100] Steve VanDeBogart Petros Efstathopoulos Eddie Kohler, Maxwell Krohn, Cli? Frey, David Ziegler, Frans Kaashoek, Robert Morris and DavidMazières. "Labels and Event Processes in the Asbestos Operating System." in: *ACM Transactions on Computer Systems* 25.4 (December 2007). issn: 07342071. doi: 10. 1145 / 1314299.1314302 (cf. p. 11 , 13 , 14).

[101] Viet Triem Valerie Tong, Andrew Clark and Ludovic Mé. "Specifying and Enforcing has INED-Grained Information Flow Policy: Model and Experiments ". in: *Mist.* 2010 (see p. 20).

[102] Clark Weissman. "Security Controls in the ADEPT-50 Time-sharing System". In: Fall Joint Computer Conference (AFIPS 1969). Las Vegas, NV, USA: ACM, Nov. 1969, p.. 119-133. doi: 10.1145 / 1478559.1478574 (cf. p. 5 - 7).

[103] Thomas Witkowski, Nicolas White, Daniel Kroening and Georg Weissenba- expensive. "Model Checking Concurrent Linux Device Drivers". In: IEEE / ACM International Conference on Automated Software Engineering (ASE 2007). Atlanta, GA, USA: ACM, 2007, p. 501-504. isbn: 978-1-59593-882-4. doi: 10.1145 / 1321631.1321719 (cf. p. 32).

[104] Chris Wright, Crispin Cowan, James Morris, Stephen and Greg Smalley Kroah-Hartman. "Linux Security Modules Framework". In Ottawa Linux Symposium. Ottawa, Ontario, Canada, 2002 (see p. 19 , 73).

[105] Chris Wright, Crispin Cowan, Stephen Smalley, James Morris and Greg Kroah-Hartman. "Linux Security Modules: General Security Support for the Linux Kernel". In: USENIX Security Symposium (USENIX 2002). San Francisco, CA, USA: USENIX Association, 2002, p. 17-31. ISBN: 1-931971-00-5 (cf. p. 19 , 27 , 73).

[106] Kwong Lok Yan and Heng Yin. "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Dynamic Views for Android Malware Analysis ". In: USENIX Security Symposium (USENIX 2012). Bellevue, WA, USA: USENIX Association, August 2012, p. 29-29 (see p. 12 , 22).

[107] Yang Junfeng, Ted Kremenek, Yichen Xie and Dawson Engler. "MECA: year extensible, expressive language and system for statically checking security properties ". In ACM Conference on Computer and Communications Security (CCS 2003). Washington DC, USA: ACM, Oct. 2003, p.. 321-334 (p. 29).

[108] Heng Yin, Dawn Song, Egele Manual, Christopher Kruegel and Engin Kirda. "Panorama Capturing System-wide Information Flow for Malware Detection and Analysis." in: *ACM Conference on Computer and Communications Security.* Alexandria, Virginia, USA: ACM, 2007, p. 116-127. isbn: 978-1-59593-703-2. doi: 10.1145 / 1315245.1315261 (cf. p. 11 , 12).

[109] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler and David Mazières. "Making Explicit Information Flow in HiSTAR". In: Symposium is Operating Systems Design and Implementation (OSDI 2006). Seattle, WA, USA: USENIX Association, Nov. 2006, p.. 263-278. ISBN: 1-931971-47-1 (cf. p. 11 , 13 , 16 , 140).

[110] Nickolai Zeldovich, Silas Boyd-Wickizer and David Mazières. "Securing Distributed Systems with FlowControl information. "In: *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation.* 8. T. Berkeley, CA, USA: USENIX Association, 2008, p. 293-308 (p. 11).

[111] Nickolai Zeldovich, Hari Kannan Michael Dalton and Christos Kozyrakis. "Hardware Enforcement of Application Security Policies Using Tagged Memory". In: USENIX Conference on Operating Systems Design and Imple- mentation (IDMC 2008). San Diego, CA, USA: USENIX Association, d. 2008 p. 225-240 (p. 11).

[112] Xiaolan Zhang, Anthony Edwards and Trent Jaeger. "Using CQUAL for Static Analysis of Authorization Hook Placement ". In: USENIX Security Symposium (USENIX 2002). San Francisco, CA, USA: USENIX Association, August 2002, p. 33- 48. isbn: 1-931971-00-5 (see p. 32).

[113] Jacob Zimmermann. "Intrusion detection set by the policy by Control greeting references. " Thesis doct. University of Rennes 1, December 16 2003 (see p. 20).

[114] Jacob Zimmermann, Ludovic Mé and Christophe Bidan. "An Improved refer- Reference Flow Control Model for Policy-Based Intrusion Detection. " in: *Pro- ceedings of the 8th European Symposium on Research in Computer Security (ESORICS).* Oct. 2003 (see p. 20).

[115] Jacob Zimmermann, Ludovic Mé and Christophe Bidan. "Experimenting with a Policy-Based HIDS Based on an Information Flow Control Model ". in: *Proceedings of the Annual Computer Security Applications Conference (CRGBA).* Dec. 2003 (see p. 21).

the author of publications associated with the contributions of this thesis

International conferences

[34] Lawrence Georget, Mathieu Jaume, Piolle William Frederick Tronel and Valérie Viet Triem Tong. "Information Flow Tracking System for Linux Handling Concurrent Calls and Shared Memory." in: *Software Engineering and Formal Methods*. Software Engineering and Formal Methods (MPAC 2017). Ed. Alessandro Cimatti and Marjan Sirjani. T. 10469. Lecture Notes in Computer Science. Trento, Italy: Springer, Cham, Switzerland, September 4, 2017,. p. 1-16. isbn: 978-3-319-66196-4. doi: 10.1007 / 978-3-319-66197-1_1 (See p. 148).

[36] Laurent Georget, Mathieu Jaume Piolle Guillaume, Frédéric and Tronel Valérie Viet Triem Tong. "Verifying the Reliability of Operating System- Level Information Flow Control Systems in Linux". In: FME Workshop on Formal Methods in Software Engineering (formalized in 2017). Buenos Aires, Argentina: IEEE Press, May 2017, p. 10-16. isbn: 978-1-5386-0422-9. doi: 10.1109 / FormaliSE.2017..1 (cf. p. 126).

[37] Laurent Georget, Frederic Tronel and Valérie Viet Triem Tong. "Kayrebt: An Activity Diagram Extraction and Visualization Toolset Designed for the Linux codebase. " In: IEEE Working Conference on Software Visualization (VISSOFT 2015). Bremen, Germany: IEEE, September 2015, p.. 170-174. doi: 10. 1109 / VISSOFT.2015.7332431 (cf. p. 71).

French Conference proceedings

[35] Laurent Georget, Mathieu Jaume Piolle Guillaume, Frédéric and Tronel Valérie Viet Triem Tong. "Monitoring? Ow proper information on Linux". In: Formal Approaches in Software Development Assistance (AFADL 2017). Ed. Akram Idani and Nikolai Kosmatov. Montpellier, France, in June 2017.

Table of? Gures

List of paintings

List of code snippets

Glossary

daemon Program running in the background and executing tasks of adminis-
tration continuously. 20

rmware Code provided by a device manufacturer material to be loaded
on the device initialization. It differs from the pilot in that it is absolutely necessary to Operates in the device and runs on celui-
This, and not the operating system. This is typically the microcode-programmable integrated components. 37

initramfs System? Kernel loading files, mini-operating system
executed by the computer system startup files to mount the root system, and unzip and start the real core. 37

patch Patch,? Shit posing as a list of di? Erence between a set
of? source code files and the same? le "corrected" for example, to add functionality or eliminate *bug*. The kernel development is organized around these *patches:* Developers wishing to offer their contribution to the kernel do as *patches* posted a list of di? usion in which they are discussed, discussed,
reworked and? nally accepted or rejected by the maintainer in charge of the relevant part of the core. 37 , 126

Process Identi? Er Identi? Ing what in reality is a task for the Linux kernel,
and which corresponds to a rather *thread* from the perspective of the user space. The name remains the time when Linux was not managing the *multithreading* in the nucleus. 190

Thread Group identi? Er Identi? Ant group *threads* , what is appoint
most common "process". 190

thread Wire execution smallest entity that can be scheduled by the kernel to
execute code. A process consists of several *threads* sharing the same signal handler (and for the sake of convenience the same address space). 10 , 16 , 18 - 21 , 37 , 40 - 45 , 47 , 55 , 58 , 59 , 68 , 74 , 75 , 80 , 126 , 189

vanilla The Linux kernel *vanilla* refers to the core "o? Heaven," as granted by Linux
Torvalds, maintainer o? Sky, and his team, without modi? Cation provided by a third party. The name comes from what some people say that natural food is "vanilla" even when it does not contain, for some mysterious reason. 21 , 75 , 152 , 153

API Application Programming Interface. 13 , 37 , 49 , 67 , *See :* Application Programming
Interface

Application Programming Interface application programming interface, mo-
dule a library, a web service, an operating system, etc.

developed, documented and maintained in order to allow a use features in the program. 189

Decentralized Information Flow Control Control greeting information decen-
ized, where each user can decide at least part of the policy to be applied to the data it owns in the system. 12 , 190

Department of Defense Equivalent unien states the Ministry of Defense. 5 , 190

DIFC Decentralized Information Flow Control. 12 , 15 , 17 , 22 , *See :* Decentralized Flow Control Information

DoD Department of Defense. 5 , 6 , 14 , *See :* Department of Defense

GCC *GNU Compilers Collection [*89]. 30 , 32 , 57 , 58 , 60 - 63 , 65 , 66 , 70 , 72 , 82 , 83 , 87 - 89 , 92 , 95 , 122 , 123 , 126 , 151

Inter-Process Communication communication channel that two or process
more can share to aChange data and collaborate, such as pipes (*pipes),* network sockets, etc. 13 , 49 , 190

IP *Internet Protocol.* 11 , 53 - 55

CPI Inter-Process Communication. 13 , 17 , 21 , 22 , 37 , 44 , 49 , 51 , 53 , 78 , 160 , *See :* Inter-Process Communication

Linux Security Modules Linux Security Modules. Refers to both a *framework*
designed to facilitate the development and integration of security modules in the Linux kernel and the modules themselves. 2 , 190

LSM Linux Security Modules. 2 , 3 , 19 - 21 , 27 , 28 , 32 , 33 , 37 - 41 , 45 , 46 , 54 , 57 , 67 , 73 - 75 , 77 , 78 , 80 - 87 , 104 , 117 , 122 - 128 , 130 , 136 , 139 , 144 - 146 , 148 , 150 - 153 , 155 , 165 , *See :* Linux Security Modules

PID *Process Identi? Er.* 44 , *See : Process Identi? Er*

Portable Operating System Interface Standard originally published by IEEE, and not
The Open Group, standardizing interfaces and basic utilities as an operating system should provide to ensure their compatibility and "UNIXité".
49 , 190

POSIX Portable Operating System Interface. 49 , 51 - 53 , *See :* Portable Operating System Interface

Remote Procedure Call Remote Function Call. Mechanism for a
process to call a function of a program running in a separate process. 18 , 190

CPP Remote Procedure Call. 18 , *See :* Remote Procedure Call

TGID *Thread Group IDenti•er. Voir : Thread Group IDenti•er*

treillis Un treillis est un ensemble équipé d'une relation d'ordre tel que toute paire
d'éléments admet un unique plus petit élément supérieur et un unique plus grand élément inférieur.

Si l'ensemble est •ni, le treillis également, et il existe alors un unique élément minimal et un unique élément maximal. On note couramment $\langle E,$
$, u, t, >, \perp \rangle$ avec :

— *E :* l'ensemble considéré ;

— : l'ordre (généralement partiel, sinon le treillis est dégénéré) ;

— *u :* l'opération *join,* donnant le plus petit élément supérieur de deux élé- ments ;

— *t :* l'opération *meet,* donnant le plus grand élément inférieur à deux élé- ments ;

— > : l'élément maximal de *E (* si *E* est ▪ni) ;

— ⊥ *:* l'élément minimal de *E (* si *E* est ▪ni).

7

UTS *UNIX Time-Sharing.* 44