La procrastination structurée

Andrew Bedford

Avril 2018

Plan

- Qu'est-ce que c'est?
- Projets que j'ai développés en utilisant cette technique

Problème

- Travailler sur un même projet pendant une longue période de temps
- Baisse de motivation est inévitable
- Réaction initiale: se forcer à travailler dessus
- Mes observations
  - Peu productif
  - Baisse de motivation
  - Fatigue
  - :(

La procrastination structurée

- Formalisée par John Perry, professeur de philosophie à Stanford
- "The Art of Procrastination" / structuredprocrastination.com
- Idée:
  1. (procrastination) Quand on a pas le goût de travailler sur quelque chose, procrastiner en travaillant sur quelque chose d'autre
  2. (structurée) Organiser nos tâches de façon à se donner le goût de travailler sur ce qu'on doit faire
- Optimisation: Minimiser le temps qu'on passe à travailler sur quelque chose quand on est pas motivé, tout en maximisant notre productivité

La procrastination structurée

TODO list

- Suggère de maintenir une TODO list avec les choses qu'on a à faire, et qu'on aimerait faire (en ordre d'importance)
- Permet de nous donner des alternatives
- Pour nous donner le goût de travailler sur ce qu'on a pas le goût de travailler, il suffit juste de bien choisir les tâches au haut de la liste
- TODO: A, B, C, D, E
- TODO: α, A, B, C, D, E
- Tâche idéale au haut de la liste: semble importante/urgente, mais ne l'est pas vraiment
- Erreur possible: Essayer de minimiser le nombre de choses à faire dans la liste; on veut plus grand choix de choses à faire

Projets

1. Ott-IFC
2. Coqatoo
3. Library-Visualizer

Ott-IFC

Situation

- Recherche porte sur les mécanismes de contrôle de flots d'information
- Objectif: Permettre à l'utilisateur de contrôler ce qu'une application peut faire avec son information
- Doctorat: Planifié implémenter un mécanisme pour Android
- Manipuler du bytecode/assembleur n'est pas très plaisant
- Découvert que quelqu'un déjà fait quelque chose de similaire

Ott-IFC

- Un outil capable d'automatiquement générer la spécification d'un mécanisme à partir de la spécification d'un langage de programmation (i.e., syntaxe et sémantique)
- Applique des règles de ré-écriture sur la sémantique du langage pour générer un moniteur

Ott-IFC

Entrée

```
<a, m, o> || <n, m, o>
------------------------------------ :: assign
<x := a, m, o> || <skip, m[x |-> n], o>
```

Ott-IFC

Sortie

```
<a, m, o> || <n, m, o>
E |- a :  la
------------------------------------- :: assign
<x := a, m, o, pc, E> ||
<skip, m[x |-> n], o, pc, E[x |-> pc |_| la]>
```

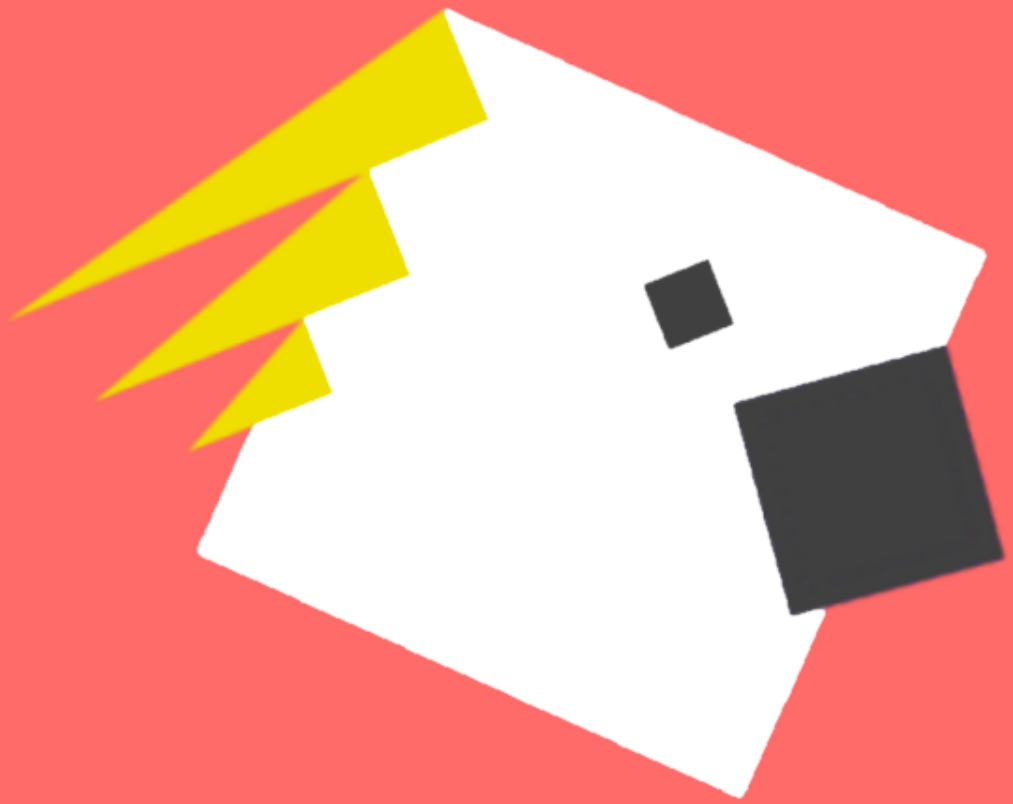
- Peut-être ensuite exportée vers LaTeX, Coq, Isabelle/HOL, Ocaml et d'autres langages

Ott-IFC

Limitations

- Requiert que la spécification du langage suit un certain format
- Support est limité à des langages impératifs simples pour le moment
- Testé sur:
  - un langage avec des if/while
  - un language avec des exceptions (try-catch/throw)
  - un language avec des locally-scoped variables (let x := e in)
- But: "compiler compiler" du monde de la sécurité ;)

Coqatoo



Situation

- Important de prouver que nos algorithmes sont corrects
- Assistants de preuves comme Coq peuvent aider avec cette tâche
- Permet d'écrire des preuves en écrivant du code; parfait pour les informaticiens!
- Un désavantage est que les preuves sont parfois difficiles à lire et à comprendre

Coqatoo

Entrée

```
Lemma conj_imp_equiv : forall P Q R:Prop,
  (P /\ Q -> R) <-> (P -> Q -> R).
Proof.
  intros. split. intros H HP HQ. apply H. apply conj.
    assumption. assumption. intros H HPQ. inversion
    HPQ. apply H. assumption. assumption.
Qed.
```

Coqatoo

Sortie (–mode text)

- Pour adresser ce problème, j'ai créé un outil (Coqatoo) qui est capable de générer des versions en langue naturelle de preuves écrites en Coq

```
Given any P, Q and R : Prop. Let us show that (P /\ Q -> R) <-> (P -> Q -> R) is
      true.
- Case (P /\ Q -> R) -> P -> Q -> R:
    Suppose that P, Q, P /\ Q -> R are true. Let us show that R is true.
    By our hypothesis P /\ Q -> R, we know that R is true if P /\ Q are true.
    -- Case P:
       True, because it is one of our assumptions.
    -- Case Q:
       True, because it is one of our assumptions.
- Case (P -> Q -> R) -> P /\ Q -> R:
    Suppose that P /\ Q, P -> Q -> R are true. Let us show that R is true.
    By our hypothesis P -> Q -> R, we know that R is true if P, Q are true.
    -- Case P:
       True, because it is one of our assumptions.
    -- Case Q:
       True, because it is one of our assumptions.
```

Coqatoo

Sortie (–mode coq)

```
Lemma conj_imp_equiv : forall P Q R:Prop, (P /\ Q -> R) <-> (P -> Q -> R).
Proof.
  (* Given any P, Q, R : Prop. Let us show that (P /\ Q -> R) <-> (P -> Q -> R)
     is true. *) intros.
  split.
  - (* Case (P /\ Q -> R) -> P -> Q -> R: *)
    (* Suppose that P, Q and P /\ Q -> R are true. Let us show that R is true.
     *) intros H HP HQ.
    (* By our hypothesis P /\ Q -> R, we know that R is true if P /\ Q  is true.
      *) apply H.
    apply conj.
    -- (* Case P: *)
       (* True, because it is one of our assumptions. *) assumption.
    -- (* Case Q: *)
       (* True, because it is one of our assumptions. *) assumption.
  - (* Case (P -> Q -> R) -> P /\ Q -> R: *)
    (* Suppose that P /\ Q and P -> Q -> R are true. Let us show that R is true.
      *) intros H HPQ.
    (* By inversion on P /\ Q, we know that P, Q are also true. *) inversion HPQ
     .
    (* By our hypothesis P -> Q -> R, we know that R is true if P and Q are true
     . *) apply H.
    -- (* Case P: *)
       (* True, because it is one of our assumptions. *) assumption.
    -- (* Case Q: *)
       (* True, because it is one of our assumptions. *) assumption.
Qed.
```

Lemma

(conj_imp_equiv) $\forall P, Q, R : Prop, (P \wedge Q \Rightarrow R) \Leftrightarrow (P \Rightarrow Q \Rightarrow R)$

Proof.

Given any P, Q *and* $R : Prop$. Let us show that $(P \wedge Q \Rightarrow R) \Leftrightarrow (P \Rightarrow Q \Rightarrow R)$ is true.

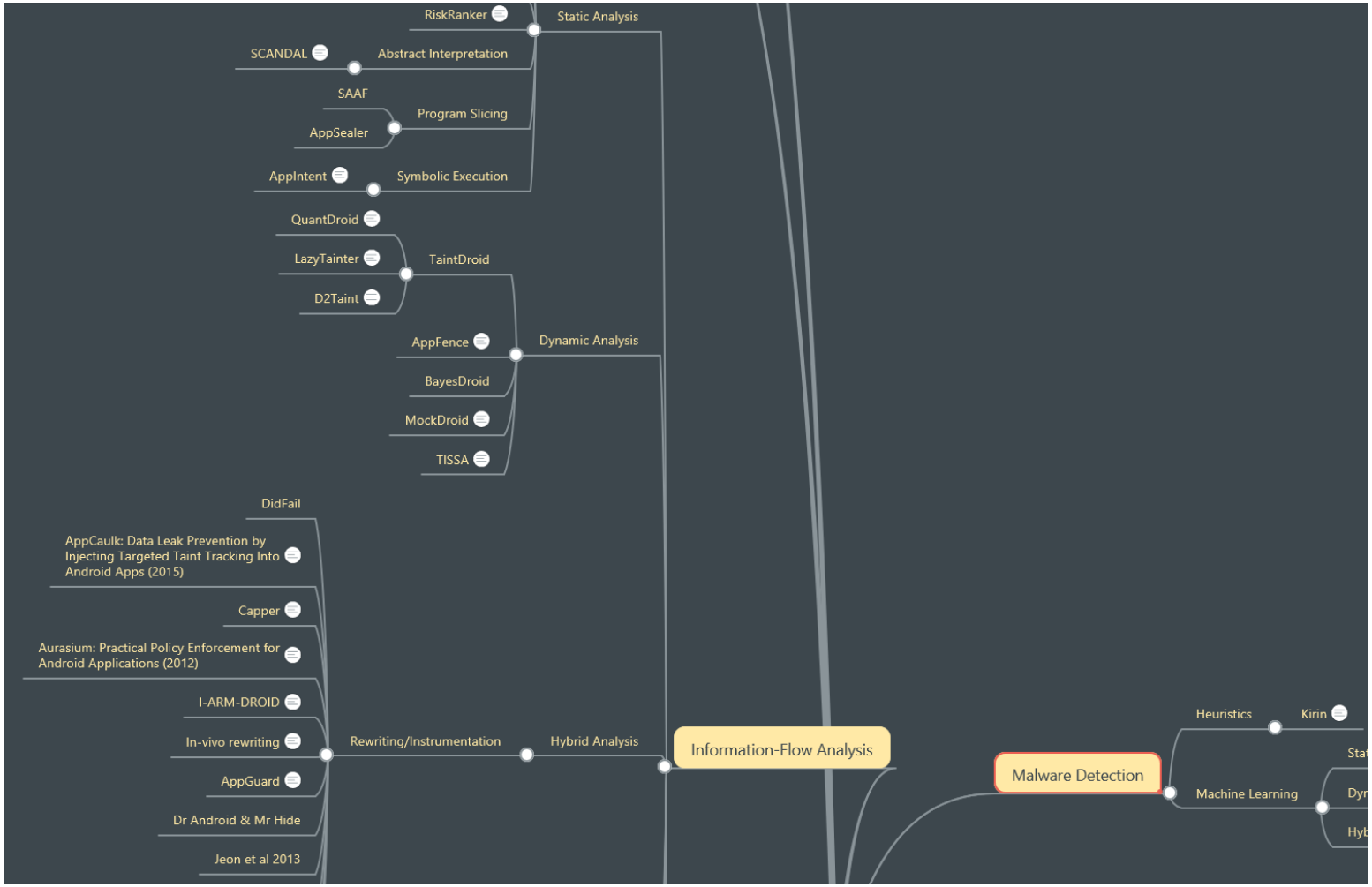
- Case $(P \wedge Q \Rightarrow R) \Rightarrow P \Rightarrow Q \Rightarrow R$:
  Suppose that $P, Q, P \wedge Q \Rightarrow R$ are true. Let us show that R is true.
  By our hypothesis $P \wedge Q \Rightarrow R$, we know that R is true if $P \wedge Q$ are true.
  – Case P:
    True, because it is one of our assumptions.
  – Case Q:
    True, because it is one of our assumptions.
- Case $(P \Rightarrow Q \Rightarrow R) \Rightarrow P \wedge Q \Rightarrow R$:
  Suppose that $P \wedge Q, P \Rightarrow Q \Rightarrow R$ are true. Let us show that R is true.
  By our hypothesis $P \Rightarrow Q \Rightarrow R$, we know that R is true if P, Q are true.
  – Case P:
    True, because it is one of our assumptions.
  – Case Q:
    True, because it is one of our assumptions.

□

Library-Visualizer

Situation

- Écrire un état de l'art pour un article



Situation

ings of the 36th International Conference on Software Engineering, Hyderabad, India, 2014.

22. Pandita R, Xiao X, Yang W, Enck W, Xie T. Whyper: towards automating risk assessment of mobile applications. In *Usenix Security*, Vol. 13, USENIX Security Symposium: Washington, DC, USA, 2013; 527–542.

6. RELATED WORK

This work is related to the approaches that use machine learning to identify Android malware, as well as the ones that leverage information flow analysis to detect sensitive data leakages.

6.1. Android malware detection

Android malware detection has received a lot of attentions. Several techniques focus on checking the *actual* behavior against the *claimed* behavior of the application. These techniques [21–23] mostly use the textual description to help describe the actual behavior of application. RISKPATH instead only requires the package files of applications and therefore can easily be applied to detect applications whose textual descriptions are difficult to obtain.

Machine learning techniques have been used by several approaches [24–29] to detect malware.

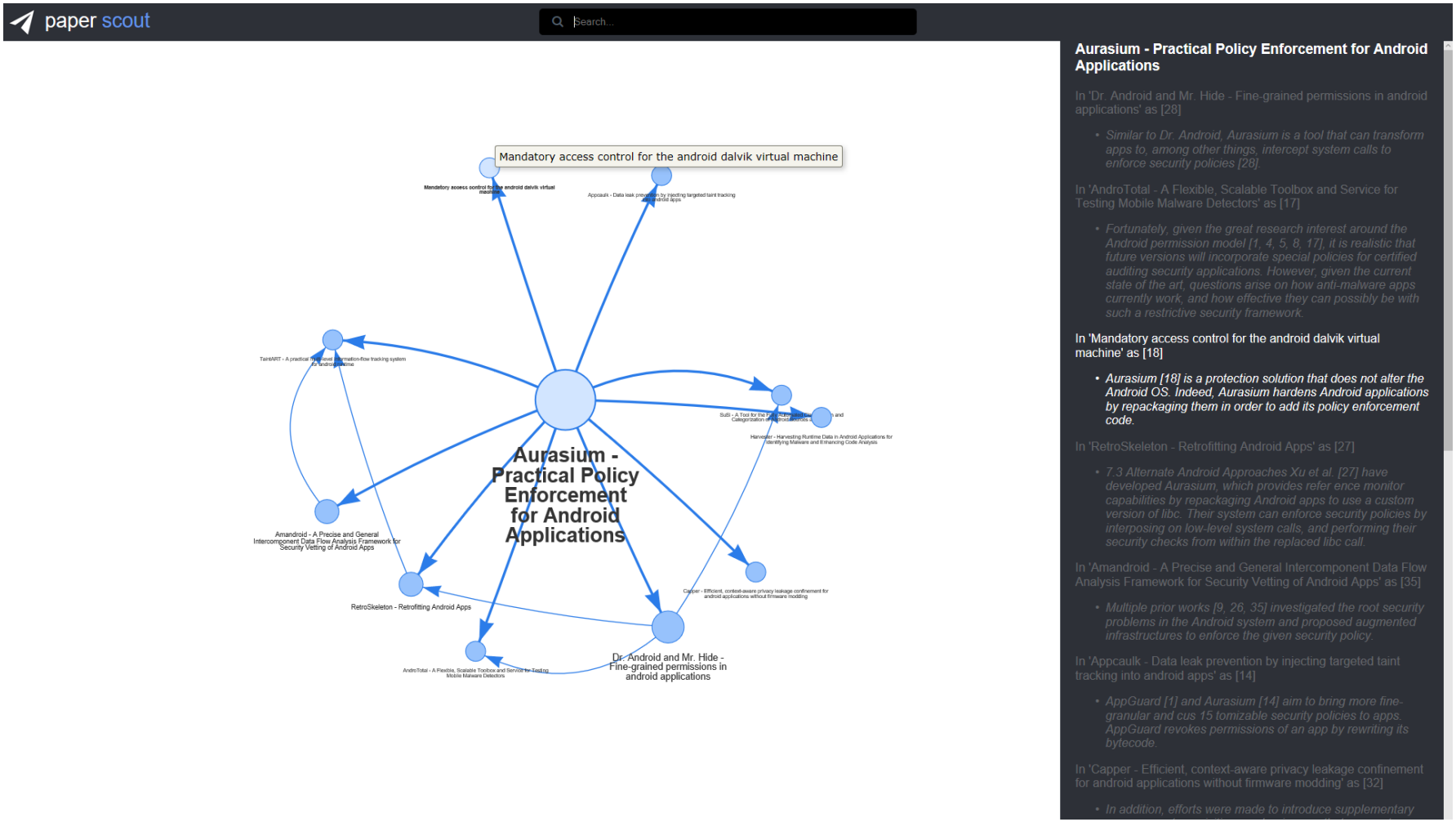
ence, TRUST, Vienna, Austria, 2012; 291–307.

32. Klieber W, Flynn L, Bhosale A, Jia L, Bauer L. Android taint flow analysis for app sets. *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*, ACM: Edinburgh, UK, 2014; 1–6.

sensitive, object-sensitive, and lifecycle-aware taint analysis for Android apps. It tracks the flow of taint data from sensitive source methods to sink methods. If an Android application transmits tainted data from predefined source methods to sink methods, then this sensitive data leakage in this app will be reported. Prior work like AndroidLeaks [31] also uses static analysis to detect intra-component sensitive data leakages in Android apps but provides less precise results. Inter-component information flows have been detected by several methods [32–34]. DidFail [32] focuses on inter-component leakages between activities through Intent. IccTa [33] reduces the inter-component information flow problems to intra-component information flow problems and leverages FlowDroid to perform inter-component information flow analysis on instrumenting Android apps.

Other techniques focus on using dynamic information flow analysis to detect sensitive data leakages. TaintDroid [35] is one of the most sophisticated dynamic taint-tracking approach on detecting sensitive data leakages within Android apps. It modified the Android Dalvik

Library-Visualizer



Library-Visualizer

CERMINE et OpenNLP

Conclusion

- Procrastination structurée: permis de varier le travail et éviter les baisses de motivation
- Explorer de nouveaux sujets et mené à de nouvelles idées
- Suggère de maintenir un journal de recherche
- A mené au développement d'outils qui sont des améliorations par rapport à l'état de l'art
- :)

GitHub

github.com/andrew-bedford